ION™ (IDL On the Net) Guide

ION Version 1.1
February, 1998 Edition

Restricted Rights Notice

The ION™ software program and the accompanying procedures, functions, and documentation described herein are sold under license agreement. Their use, duplication, and disclosure are subject to the restrictions stated in the license agreement.

Limitation of Warranty

Research Systems, Inc. makes no warranties, either express or implied, as to any matter not expressly set forth in the license agreement, including without limitation the condition of the software, merchantability, or fitness for any particular purpose.

Research Systems, Inc. shall not be liable for any direct, consequential, or other damages suffered by the Licensee or any others resulting from use of the ION software package or its documentation.

Permission to Reproduce this Manual

Purchasers of ION licenses are given limited permission to reproduce this manual provided such copies are for their use only and are not sold or distributed to third parties. All such copies must contain the title page and this notice page in their entirety.

Acknowledgments

ION™ documentation is printed on recycled paper. Our paper has a minimum 20% post-consumer waste content and meets all EPA guidelines.

Contents

Chapter 1

Overview

The following topics are covered in this chapter:

What is ION?

IDL On the Net (ION) is a sophisticated (yet simple to implement) system that brings the power of IDL to the Internet. ION uses the latest Java and Internet technology to deliver efficient data analysis and visualization capabilities to World Wide Web client applications. ION is ideal for organizations that have shared data that needs to be accessed and visualized by a wide variety of users. ION can be configured as part of a public Web server, a proprietary intranet server, or as both at the same time.

ION allows access to IDL from virtually any computer in the world. Updating and maintaining ION is simple, since the product resides only on the server. Applets are sent to clients over the Web, as needed.

ION is built on IDL, which means that existing IDL programs can be easily converted for Internet access. ION is simple to use, requiring only basic knowledge of IDL and HTML. Pre-built Java applets are included, so there is no need to become a Java programmer to use ION. However, full Java integration is provided for those who wish to develop custom Java applications that work in concert with ION.

Note Only Unix and Windows NT versions of the ION Server are available. (See "Configuring the ION Server" on page 51 for details.) Since Java is a cross-platform language, however, there are no platform limitations on ION client applications.

ION Limitations

Server Limitations

The ION version 1.1 Tunnel Broker does not work through network firewalls that do not use the SOCKS protocol. (See "The ION HTTP Tunnel Broker" on page 58 for details on the Tunnel Broker.)

IDL Limitations

ION does not allow access to *all* IDL features. Specifically, ION does not let you use:

- IDL Widgets
- IDL Object Graphics
- the line continuation character ("$")

All of IDL's analytical routines and all of the IDL Direct Graphics routines are available, subject to the constraints imposed by the ION security mechanism. (See "Command Security" on page 52 for more on ION's security mechanism.)

ION Components

The ION package consists of the following components:

ION Server

The ION Server is a program that manages communication between an ION client application (either a Java Applet running in a Web browser or a stand-alone Java application) and IDL. The ION Server translates requests from ION clients into commands that can be processed by IDL, and then passes output from IDL back to the client for display. The ION Server is discussed in detail in "Configuring the ION Server" on page 51.

ION Daemon

The ION Daemon is a program that makes the initial connection between an ION client and the ION Server. The ION Daemon "watches" a specific port on the ION Server's host computer. When the daemon receives a request for connection, it performs basic security screening before connecting the ION client to the ION Server. The ION Daemon is discussed in detail in "Configuring the ION Server" on page 51.

ION HTTP Tunnel Broker

The ION HTTP Tunnel Broker is a program that allows ION client applets (running in World Wide Web browsers) located behind network firewalls to communicate with an ION Server on the other side of the firewall. The Tunnel Broker is discussed in detail in "The ION HTTP Tunnel Broker" on page 58.

Pre-Built ION Client Applets

The ION package includes a set of pre-built Java applets. The pre-built applets allow you to begin using ION immediately, without the need to write Java code. See "Using ION's Java Applets"on page 11 for details.

ION Graphics Java Classes

The ION Graphics Java classes provide a simple, straightforward interface that allows you to create ION client applets and applications quickly and easily. While using the ION Graphics classes does require that you write Java code, the classes do handle most of the details of writing applications to interact with IDL seamlessly. See "ION Java Classes"on page 21 for details.

ION Low-Level Java Classes

The ION Low-Level Java classes are the backbone of the ION system; they provide the tools a professional Java programmer needs to create robust applications to interact with IDL. The ION Graphics Java classes and the ION pre-built applets are both built directly from the ION Low-Level classes. See "ION Low-level Java Classes"on page 29 for details

Documentation

ION documentation is provided inprint, HTML, and Adobe Acrobat (PDF) formats. HTML documentation is included in your ION installation in the `docs` subdirectory; open the file `ion.html` to navigate to the rest of the documentation. The Adobe Acrobat version of the documentation is located in the file `ion.pdf` in the help subdirectory. Note that version 3 or later of the Adobe Acrobat viewer application is required to view the `.pdf` file. You can download the Adobe Acrobat viewer free directly from Adobe.

Skills Necessary to use ION

ION is designed to make it easy for you to create interactive Web pages or Internet/Intranet applications that use IDL. In order to use ION effectively, and depending on your objective, you will need one or more of the following:

Familiarity with IDL

ION is designed to interact with IDL. To use ION, you will need to be familiar with IDL's basic command syntax and features. For more information about IDL, consult your IDL documentation.

Familiarity with HTML

In order to include ION Java applets in Web pages, you will need to be familiar with the HyperText Markup Language (HTML). There are numerous books at all skill levels available if you wish to learn HTML; be sure to pick a reference that discusses HTML version 3 or later. One book we have found useful is the *HTML Sourcebook, Third Edition* by Ian S. Graham (Wiley Computer Publishing, New York, 1997. ISBN 0-471-17575-7). You may also wish to visit the World Wide Web Consortium's web site at `www.w3.org` for details about the Web, HTML, and other topics of interest.

Familiarity with using Java applets

Java applets are embedded into Web pages using standard HTML code and the APPLET tag. Consult your HTML documentation or visit `java.sun.com/applets` for information about using applets in HTML pages.

Familiarity with building Java applets

If you wish to build your own applications or applets to use the ION Server, you will need to be familiar with Java programming concepts. Again, there are numerous reference books available describing Java and Java programming. Some that we have found useful are:

The Java Tutorial by Mary Campione and Kathy Walrath (Addison-Wesley, Reading, MA, 1996. ISBN 0-201-63454-6)

The Java Class Libraries by Patrick Chan and Rosanna Lee (Addison-Wesley, Reading, MA, 1997. ISBN 0-201-63458-9)

Java in a Nutshell, by David Flanagan (O'Reilly & Associates, Sebastopol, CA, 1996. ISBN 1-56592-183-6)

You can download the Java Developer's Kit directly (and find current information about and reference material pertaining to Java) from `java.sun.com`.

Familiarity with Web Servers

Even if you do not maintain the World Wide Web server at your site, you should be aware of the configuration details. You will need to know where files should be located for access

by the server, what file permissions are necessary, and any other site-specific details that apply to publishing HTML pages on the World Wide Web.

Typographical Conventions

The following typographical conventions are used in this book:

- UPPER CASE
  IDL functions, procedures, and keywords are displayed in UPPER CASE type. For example, the calling sequence for an IDL procedure looks like this:

  CONTOUR, *Z [, X, Y]*

- Mixed Case
  ION object class and method names are displayed in Mixed Case type. Unlike IDL, the Java language is case-sensitive; names of ION Java methods and classes must be entered with the same capitalization as shown in this reference section.

- *Italic type*
  Arguments to ION procedures and functions — data or variables you must provide — are displayed in italic type.

- Square brackets ([ ])
  Square brackets used in calling sequences indicate that the enclosed arguments are optional. Do not type the brackets. In the above CONTOUR example, *X* and *Y* are optional arguments. Square brackets are also used to specify array elements.

- `Courier type`
  Names of ION classes and methods are displayed in `courier` type. Syntax descriptions are shown in **`courier bold`** and ***`bold italic`***. Examples are shown in `courier`. Uniform Resource Locators (URLs) are also shown in `courier`.

About the Example Code

Example code illustrating ION features is included in the installed ION distribution. You will find example HTML files located in subdirectories of the examples directory in your installed ION distribution. Browse through the example code starting with the file `examples.html` in the `examples` directory. Many of the examples allow you to view the Java source for the example within your browser. The HTML version of the *ION User's Guide* (this document) includes links to the installed example HTML files.

The raw Java source files for the example ION classes are included in the `src` subdirectory of the `classes` directory. Also included in the src subdirectory are a number of IDL .pro files that are called by the ION demonstration applets. In order to use the ION demos on your own system, you must add the directory `RSI_DIR/ion_1.1/classes/src` to IDL's path. See "Location of IDL .pro Files" on page 223.

Note For the examples to function properly, you must have the ION Server running on your machine. If you do not yet have the ION Server running on your system, visit Research Systems' ION web site and view example code there.

Chapter 2

Installation and Configuration

The following topics are covered in this chapter:

This chapter discusses the process of installing ION on your system. See "Configuring the ION Server" on page 51 for details on setting up the ION Server once it is installed.

The ION installation process is very similar to the IDL installation process. The ION directory tree `ion_release` (where *release* is the version number of the ION release you are installing) is installed in your Research Systems product directory (denoted here as *RSI_DIR*). In addition, a copy of IDL is installed in the ION directory tree, in the `idl_5` subdirectory (Unix) or the `idl50` subdirectory (Windows).

IDL must be installed in the ION tree even if you have another IDL installation on the same host machine, for two reasons:

1. Because ION loads IDL features dynamically, each version of ION depends on a specific version of IDL. ION version 1.1 relies on IDL version 5.0.3. ION may not work properly if a different version of IDL is substituted.
2. ION expects to find the IDL libraries in a well-defined location. ION will not work if it cannot find the IDL library.

   The ION Server communicates with a copy of IDL installed in the ION hierarchy. If ION is installed in `/usr/local/rsi/ion_1.1`, IDL is expected to be located in the directory `/usr/local/rsi/ion_1.1/idl_5`.

There are slight differences in the installation procedures depending on whether you have downloaded the ION distribution from the Internet or have received an ION CD-ROM.

Installing ION under Unix

If you downloaded ION from a World Wide Web Page

If you have downloaded ION from a World Wide Web page, you will need to use the Unix `tar` program to extract the installation files. Place the tar file downloaded from the Web page in the RSI directory (usually `/usr/local/rsi`) and execute the command

```
tar xvf ion11_<os_name>.tar
```

where *<os_name>* is the name of the platform you have downloaded ION for. Read the file `README.TXT` for information that was too late to include in the main documentation. Installation instructions are available in an Adobe Acrobat file named ion_inst.pdf.

If you downloaded ION from an FTP Server

If you have downloaded ION from an FTP server, ION is installed on your system by a shell script named `unpack`. Follow the instructions in the file `README.TXT` located in the same directory as the ION archive files to download and install ION on your system. Installation instructions are available in an Adobe Acrobat file named ion_inst.pdf.

If you have an ION CD-ROM

If you have an ION CD-ROM, consult the printed *Installation Guide*.

Installing ION under Windows

The ION installation process is the same whether you have downloaded ION from the Internet or received a CD-ROM. Simply run `setupex.exe` to run the installation program and follow the prompts. Additional installation instructions can be found in the *Installation Guide*. If you have an ION CD-ROM, you should have received a printed *Installation Guide*. If you downloaded ION from the Internet, the *Installation Guide* is available in an Adobe Acrobat file named ion_inst.pdf.

Location of ION Class Files

Java Applet security mechanisms require that the ION class files reside on the same host machine as the ION Server. During installation, ION class files are installed in the following location on the ION Server:

RSI_DIR`/ion_`*release*`/classes/`

where *RSI_DIR* is the location of the RSI directory on your system. By default, *RSI_DIR* is `/usr/local/rsi`. The ION directory is named `ion_`*release*, where *release* is the release number. For example, on Unix systems, ION release 1.1 the directory is named `ion_1.1`, and for Windows systems the directory is named `ion11`.

ION's Java class files are provided in three formats:

1. A *package* of Java class files named *IONclass*`.class`, where *IONclass* is the name of the class implemented. Java packages are structured in a directory hierarchy and named accordingly; the ION package has the name `com.rsi.ion`, which means that the class files themselves are in the `com/rsi/ion` subdirectory of the `classes` directory.
2. An uncompressed zip file named `ion_`*release*`.zip`, where *release* is the version number of the ION release. The zip file is installed in the `classes` directory.
3. A Java ARchive (jar) file named `ion_`*release*`.jar`, where *release* is the version number of the ION release. The jar file is installed in the `classes` directory.

Locating the Class Files for use by ION Applets

ION applets must have access to the ION class files in order to run. While you can use the CODEBASE attribute to specify a relative path from the location of an HTML page containing an ION applet tag to the location of the class files, it is often easier to copy the class files (or provide a symbolic link, if your system supports symbolic links) to another directory located in or near the directory containing your HTML files.

For example, suppose you have located your HTML pages in a directory named `public_html`. You may wish to place the ION package, the ION zip file, and the ION jar file in a subdirectory of `public_html` named `java`. If you then include any ION applet class files you create in the `java` directory, you could simply specify:

```
CODEBASE="./java"
```

in the `<APPLET>` tag used in your HTML page.

See the discussion of the CODEBASE attribute on page 13 for further details.

About Web Servers

ION does not include World Wide Web server software, and this document does not discuss the installation or configuration of web servers. You will need to have web server software installed and properly configured before ION will function.

The Apache Group makes source-code and compiled versions of its web server software available at no cost. Servers are available for most Unix flavors, and as of this writing (January, 1998) a Windows NT version is currently in the beta-test stage. Visit http://apache.org for details.

Chapter 3

Using ION's Java Applets

The following topics are covered in this chapter:

The ION package is designed to give you a flexible way to use IDL in a distributed, networked environment. This can mean anything from presenting graphics created with IDL on the World Wide Web to creating sophisticated client-side applications that use IDL's analytical and graphical facilities. To address the wide range of possible tasks, the ION package includes everything from pre-built Java applets to a low-level Java class library.

A Note on the Examples

You can use the examples in the chapter directly in your own web pages by specifying the appropriate host and port settings for your server, and by specifying the `CODEBASE` attribute to reflect the location of the ION class files. Before you can use the ION Java applets, however, you must have the ION Server installed, configured, and running. See "Configuring the ION Server" on page 51 for details.

What are the Pre-Built Applets?

The Java applets included with the ION package allow you to interact with the ION Server with a minimum of knowledge of or experience with the Java language. Because the applets are pre-built, you can include them in Web pages using only HTML code. This chapter discusses the pre-built applets, describes how to set up and customize each applet, and provides example code.

Using ION Applets

Use the HTML tag `<APPLET>` to include ION applets in your HTML code. Information required by the applet is provided via *attributes*, either within the `APPLET` tag itself or in one or more `PARAM` tags specified for the applet.

You can include HTML text within an applet tag, but the text will only be displayed if the Java virtual machine fails to start. You may find it useful to include something like the following:

```
<APPLET attributes>
   <!- Applet code >
   <B>Java virtual machine failed to start.
   Is Java enabled in your browser? </B>
</APPLET>
```

People with browsers that do not support Java would see the text:

Java virtual machine failed to start. Is Java enabled on your browser?

while those with browsers that do support Java would see only the applet.

Attributes Specified in the Applet Tag

In addition to attributes required by the browser, you can specify the following attributes when you create the applet:

NAME
A string containing a unique name for the applet. The string should be enclosed in double quotes marks. This attribute is required for all ION applets.

WIDTH
The width of the applet in pixels. ION uses the WIDTH attribute when creating the drawing area. This attribute is required for all ION applets.

HEIGHT
The height of the applet in pixels. ION uses the HEIGHT attribute when creating the drawing area. This attribute is required for all ION applets.

CODE
A string specifying the name of the applet class. The CODE attribute should specify *fully-qualified* class name relative to the directory in which the HTML file is located. If the CODEBASE attribute is included, the class name specified in the CODE attribute should be relative to the directory specified by CODEBASE.

For example, If you were to place an HTML file that used the IONPlotApplet in an `HTML` subdirectory of the ION directory, the CODE and CODEBASE attributes would be:

```
CODE=com.rsi.ion.IONPlotApplet.class
```

```
CODEBASE="../classes"
```

because the `IONPlotApplet.class` file is located in the `com/rsi/ion` subdirectory of the `classes` directory in the ION distribution. Similarly, if you were to place all of the Java class files necessary for your applet in the directory containing your HTML files, you could omit the CODEBASE attribute and use something like the following:

```
CODE=MyApplet.class
```

This attribute is required for all ION applets.

CODEBASE
The CODEBASE attribute is not strictly required, but is often useful. The Java class loader searches for the contents of the `classes` directory in *current directory* — that is, the directory from which the HTML page containing the applet code was loaded. If you locate the HTML page somewhere other than the *`RSI_DIR`*`/ion`*`release`*`/classes` directory, you will need to set the CODEBASE attribute to the *relative* path from the page location to the `classes` directory, or to a URL that specifies the location.

Note If the CODEBASE attribute is set equal to a URL, then the host specified by the URL can be used for ION network connections, but the host that is serving the HTML page cannot. This allows you to set up the ION Server and all of the ION class files

on a machine separate from your web server, provided you include the SERVER_NAME parameter with the same hostname as in the CODEBASE URL.

If you use this method, both the CODEBASE and SERVER_NAME attributes must refer to the same machine or Java security errors will result. In addition, the ION Server machine will still need to run a web server, but it will only be used to get the `.class` (or archive) files for the applets.

For example, if your HTML page is located in the *RSI_DIR*/ion*release*/mypages directory, you would set the CODEBASE attribute as follows:

```
CODEBASE="../classes"
```

ARCHIVE

The ARCHIVE attribute is not required, but it can speed the downloading of Java class files for browsers that support it. See "Supporting Java Archive Files" on page 39 for a discussion of Java archive files.

ALT

The ALT attribute specified a text string to be displayed if for some reason the applet cannot be loaded. The ALT attribute is not required, but consider adding something like the following to your applet description to enhance the user-friendliness of your HTML page:

```
ALT="ION Applet failed to load. Is Java enabled in your browser?"
```

Note If you include HTML-formatted text within your APPLET tag, it will be displayed only if the Java Virtual Machine fails to start. This is slightly different from the ALT attribute, which contains text to be displayed only if the Java applet fails to load.

Example

The following `APPLET` tag creates an applet of the IONGraphicApplet class, with a drawing area 100 pixels by 100 pixels, with the name "MyApplet." The HTML page containing the applet code is assumed to be located in the directory *RSI_DIR*/ion*release*/classes, so no CODEBASE attribute is included.

```
<APPLET NAME="MyApplet" WIDTH=100 HEIGHT=100
    CODE=com.rsi.ion.IONGraphicApplet.class>
<!- Other applet code >
</APPLET>
```

Parameters Specified via PARAM Tags

The HTML tag `<PARAM>` includes a `NAME` attribute and a `VALUE` attribute, each of which is enclosed in double quotes. Use the `PARAM` tag to specify further applet parameters. This section discusses parameters common to all ION applets; parameters specific to individual applets included in the ION package are discussed in the applet-specific sections below.

Connecting to the ION Server

Before IDL commands can be executed and graphics created, the ION applet must connect to the ION Server. Establish a connection by including the following connection parameters in the HTML code that creates the applet. Note that each `PARAM` tag has a `NAME` attribute and a `VALUE` attribute, each of which is enclosed in double quotes.

SERVER_NAME

Set this value of this parameter equal to the name of the computer on which the ION Server is running. The server name can be either a simple host name (i.e. `myhost`) or a fully-qualified domain name (i.e. `myhost.mycompany.com`). Java security mechanisms require that the applet be located on the same machine as the ION Server. If the server name is not provided, the host name of the machine from which the applet was loaded is used.

PORT_NUMBER

The port number of the port on the server where the ION Daemon is listening. By default, the ION Server listens to port 7085.

SERVER_DISCONNECT

Set the value of this parameter equal to "YES" if you want the applet to disconnect from the server when all commands have been processed. (Note that if more than one applet is using the connection, the connection will not be closed until all commands from all of the connected applets have been completed.) The default value is "NO".

CONNECTION_TYPE

Set the value of this parameter to specify what type of connection ION should use. The three possible values are:

- `"HTTP_CON"` — Make only HTTP connections, using the The ION HTTP Tunnel Broker.
- `"SOCK_CON"` — Make only socket connections, using only the ION Daemon.
- `"BEST_CON"` — Attempt to make a socket connection. If a socket connection is not possible, attempt to make an HTTP connection. This is the default setting.

See "The ION HTTP Tunnel Broker" on page 58 for additional details about HTTP connections.

CONNECTION_TIMEOUT

Set the value of this parameter to an integer number of seconds to wait before assuming that a socket connection has failed. If the CONNECTION_TYPE parameter is set to "BEST_CON", ION will attempt to make an HTTP connection if the timeout time expires before a socket connection is made.

HTTP_HOSTNAME

Set the value of this parameter equal to the hostname of the computer on which the ION HTTP Tunnel Broker is running.

HTTP_PORT

Set the value of this parameter equal to the port number the ION HTTP Tunnel Broker is listening to.

Example

The following connects the "MyApplet" applet to a server named "Server1", using the default port number, the defaut connection type, and specifies that the applet should not disconnect from the server when all commands have been processed:

```
<APPLET NAME="MyApplet" WIDTH=100 HEIGHT=100
    CODE=com.rsi.ion.IONGraphicApplet.class>
    <PARAM NAME="SERVER_NAME" VALUE="Server1">
    <PARAM NAME="SERVER_DISCONNECT" VALUE="NO">
<!- Other applet code >
</APPLET>
```

Using the Same Connection for Multiple Applets

Multiple ION applets can share a single connection to the ION Server. Since each open connection consumes network bandwidth, it is often efficient to let several applets share the same connection.

To specify an existing connection for a new applet, use the ION_CONNECTION_NAME parameter rather than the SERVER_NAME, PORT_NUMBER, and SERVER_DISCONNECT parameters.

Note All applets using the same connection must be loaded into the browser at the same time. In general, this means that applets that share a connection should be included in the same HTML page.

ION_CONNECTION_NAME

Set this value of this parameter equal to the name of the applet whose connection you wish to share. The applet's name is specified by the NAME attribute in the APPLET tag.

Example

The following creates a second applet named "AnotherApplet" and specifies that it share the server connection created for "MyApplet":

```
<APPLET NAME="AnotherApplet" WIDTH=100 HEIGHT=100
    CODE=com.rsi.ion.IONGraphicApplet.class>
    <PARAM NAME="ION_CONNECTION_NAME" VALUE="MyApplet">
<!- Other applet code >
</APPLET>
```

Behavior Parameters

Two behavior parameters determine how an applet responds to certain user actions. The two behaviors currently supported by all ION applets allow the applets to display debug

information and link to other HTML pages. Use the following parameters to alter the behavior of pre-built applets:

DEBUG_MODE
If the value of this parameter is set to "YES", holding down the shift key and clicking the mouse in the applet drawing area displays a window containing the IDL commands and server responses associated with the applet's connection. If more than one applet is connected to the connection, the information for all applets is displayed. If the main connection has `DEBUG_MODE` set to "NO" (or not specified), but an applet connected to it has `DEBUG_MODE` turned on, debug will be turned on for the entire connection. The default value is "NO."

LINK_URL
Set the value of this parameter to a URL that will be loaded if the user clicks in the applet area. The switch to the linked URL happens before any mouse events are passed to the server. This option should not be used with ION applets running IDL routines that accept mouse input.

Example

The following specifies that the "MyApplet" applet will display debug information and will link to the Research Systems web page if the user clicks in the applet drawing area:

```
<APPLET NAME="MyApplet" WIDTH=100 HEIGHT=100
    CODE=com.rsi.ion.IONGraphicApplet.class>
    <PARAM NAME="SERVER_NAME" VALUE="Server1">
    <PARAM NAME="SERVER_DISCONNECT" VALUE="NO">
    <PARAM NAME="DEBUG_MODE" VALUE="YES">
    <PARAM NAME="LINK_URL" VALUE="www.rsinc.com">
<!- Other applet code >
</APPLET>
```

IONGraphicApplet

The IONGraphicApplet is used to execute a series of IDL commands and display the results. Any valid IDL commands that are not explicitly excluded by the ION security mechanism (see "Command Security" on page 52) can be passed to the IONGraphicApplet for execution. Using the ION Applet parameters, the Applet can also display debug information and be used as a hyperlink to another HTML page.

The IDL commands can be sent synchronously or asynchronously. By default, each command is sent and the client *blocks* (stops accepting commands) until the command is complete. However, in some circumstances the client needs to regain control of the application immediately to be able to process user input. An example of this situation would be when a command starts an IDL routine that enters an event loop. If the command is blocking, the client will not be free to receive data from the server or provide input the server may request.

Parameters

In addition to the parameters described in "Parameters Specified via PARAM Tags" on page 14, the IONGraphicsApplet accepts the following parameters:

IDL_COMMAND_0, ..., IDL_COMMAND_*n*

The `IDL_COMMAND_*` parameters specify the IDL commands to send to the ION Server. The value of each `IDL_COMMAND` is a valid, single line IDL command (the "$" line continuation is not supported by ION). Note that commands that are explicitly excluded via the ION security mechanism are not processed.

Note Command numbers must be continuous, beginning with zero and ending with *n*.

AYSNC_COMMANDS

Set the value of this parameter to "YES" if the client should send commands asynchronously. The default value is "NO".

DECOMPOSED_COLOR

If set to "YES", the applet will treat pixel values as RGB triplets when on a true-color (24-bit or 32-bit) device. (This is the default.) If set to "NO", the applet will treat the first eight bits (the red portion) of the pixel value as an index into the current color table when displaying on a true color device. For more information on decomposed color mode, see the documentation for the DECOMPOSED keyword to the DEVICE procedure in the IDL *Reference Guide*.

Example

The following example creates an IONGraphicsApplet that connects to a server, generates some data, sets the color table, and displays the data using IDL's SHOW3 procedure. In the example, debugging mode is enabled, and the applet drawing area is a link to the Research Systems web page.

```
<APPLET NAME="CONNECTION" CODE=IONGraphicApplet.class
```

```
        WIDTH=200 HEIGHT=200>
    <! This applet connects to host KIROC, port 8084 >
    <PARAM NAME="SERVER_NAME" VALUE="KIROC">
    <PARAM NAME="PORT_NUMBER" VALUE="8084">
    <PARAM NAME="LINK_URL" VALUE="www.rsinc.com">
    <PARAM NAME="DEBUG_MODE" VALUE="YES">
    <PARAM NAME="SERVER_DISCONNECT" VALUE="YES">
    <PARAM NAME="IDL_COMMAND_0"
        VALUE="a = exp(-(shift(dist(30), 15, 15)/7)^2)">
    <PARAM NAME="IDL_COMMAND_1" VALUE="loadct, 1">
    <PARAM NAME="IDL_COMMAND_2" VALUE="show3, a">
</APPLET>
```

IONContourApplet

The IONContourApplet displays an IDL contour plot. The X, Y and Z values of the plot and any IDL Contour properties supported by ION can be set through parameters to the applet.

Note You can also create contour plots using the IONGraphicApplet, specifying the contour properties in IDL command strings. The IONContourApplet is merely a simplified way to display contour plots.

Parameters

In addition to the parameters described in "Parameters Specified via PARAM Tags" on page 14, the IONContourApplet accepts the following parameters:

X_VALUES

Set the value of this parameter equal to a valid IDL expression that evaluates to a vector or two-dimensional array specifying the X coordinates for the contour surface. If X_VALUES specifies a vector, each element specifies the X coordinate for a column in the Z_VALUES array(e.g., X[0] specifies the X coordinate for Z[0,*]). If X_VALUES specifies a two-dimensional array, each element specifies the X coordinate of the corresponding point in the Z_VALUES array.

Y_VALUES

Set the value of this parameter equal to a valid IDL expression that evaluates to a vector or two-dimensional array specifying the Y coordinates for the contour surface. If Y_VALUES specifies a vector, each element specifies the Y coordinate for a column in the Z_VALUES array(e.g., Y[0] specifies the Y coordinate for Z[0,*]). If Y_VALUES specifies a two-dimensional array, each element specifies the Y coordinate of the corresponding point in the Z_VALUES array.

Z_VALUES

Set the value of this parameter equal to a valid IDL expression that evaluates to a one- or two-dimensional array containing the values that make up the contour surface. If the X_VALUES and Y_VALUES parameters are provided, the contour is plotted as a function of the (X, Y) locations specified by their contents. Otherwise, the contour is generated as a function of the two-dimensional array index of each element of Z_VALUES.

contour_property_1, ..., contour_property_*n*

Here, `contour_property_*` is the name of a contour property supported by the IONGrContour class. Properties for the IONContourApplet reflect the capabilities implemented in keywords to the IDL CONTOUR procedure.

Note Unlike the IONGraphicApplet `IDL_COMMAND_*` parameter, the `contour_property` parameters are not numbered.

The following IDL Contour properties are supported by IONContourApplet. Refer to the IDL documentation on keywords available for use with the CONTOUR procedure for an explanation of each property:

C_ANNOTATION, C_CHARSIZE, C_COLORS, C_LABELS, C_LINESTYLE, C_ORIENTATION, C_SPACING, CLOSED, DOWNHILL, FILL, CELL_FILL, FOLLOW, IRREGULAR, LEVELS, NLEVELS, OVERPLOT, BACKGROUND, CHARSIZE, CLIP, COLOR, DATA, DEVICE, FONT, LINESTYLE, NOCLIP, NODATA, NOERASE, NORMAL, POSITION, SUBTITLE, T3D, TICKLEN, TITLE, MAX_VALUE, MIN_VALUE, NSUM, POLAR, XLOG, YNOZERO, YLOG, XCHARSIZE, YCHARSIZE, ZCHARSIZE, XGRIDSTYLE, YGRIDSTYLE, ZGRIDSTYLE, XMARGIN, YMARGIN, ZMARGIN, XMINOR, YMINOR, ZMINOR, XRANGE, YRANGE, ZRANGE, XSTYLE, YSTYLE, ZSTYLE, XTICKFORMAT, YTICKFORMAT, ZTICKFORMAT, XTICKLEN, YTICKLEN, ZTICKLEN, XTICKNAME, YTICKNAME, ZTICKNAME, XTICKS, YTICKS, ZTICKS, XTICKV, YTICKV, ZTICKV, XTITLE, YTITLE, ZTITLE, ZVALUE, ZAXIS

Example

The following example creates an IONContourApplet that connects to the same server used by the "Connection" applet defined in the IONGraphicApplet example. The applet generates some data for the Z value of the contour, and sets the "Title" property of the contour plot.

```
<APPLET NAME="CONTOUR" CODE=IONContourApplet.class
    WIDTH=200 HEIGHT=200>
    <! This applet uses the applet 'CONNECTION' to connect
        to the server>
    <PARAM NAME="ION_CONNECTION_NAME" VALUE="CONNECTION">
    <PARAM NAME="Z_VALUES" VALUE="exp(-(shift(dist(30), 15,
        15)/7)^2)">
    <PARAM NAME="TITLE" VALUE="Contour">
</APPLET>
```

Note that the example uses an IDL expression to generate the Z values for the contour. The Z values could also have been specified as an IDL array, with a statement like:

```
    <PARAM NAME="Z_VALUES"
        VALUE="[[1,2,3,4][2,3,4,5][3,4,5,6][4,5,6,7]]">
```

IONPlotApplet

The IONPlotApplet displays an IDL plot. The X and Y values of the plot and any IDL plot properties supported by ION can be set through parameters to the applet.

Note You can also create plots using the IONGraphicApplet, specifying the plot properties in IDL command strings. The IONPlotApplet is merely a simplified way to display plots.

Parameters

In addition to the parameters described in "Parameters Specified via PARAM Tags" on page 14, the IONPlotApplet accepts the following parameters:

X_VALUES

Set the value of this parameter equal to a valid IDL expression that evaluates to a vector of X data.

Y_VALUES

Set the value of this parameter equal to a valid IDL expression that evaluates to a vector of Y data. If `Y_VALUES` is not specified, the data in `X_VALUES` is plotted as a function of point number (starting at zero). If both arguments are provided, `X_VALUES` is plotted as a function of `Y_VALUES`.

plot_property_1, ..., plot_property_*n*

Here, `plot_property_*` is the name of a plot property supported by the IONGrPlot class. Properties for the IONPlotApplet reflect the capabilities implemented in keywords to the IDL PLOT procedure.

Note Unlike the IONGraphicApplet `IDL_COMMAND_*` parameter, the `plot_property` parameters are not numbered.

The following IDL Plot properties are supported by IONPlotApplet. Refer to the IDL documentation on keywords available for use with the PLOT procedure for an explanation of each property:

BACKGROUND, CHARSIZE, CLIP, COLOR, DATA, DEVICE, FONT, LINESTYLE, NOCLIP, NODATA, NOERASE, NORMAL, POSITION, PSYM, SUBTITLE, SYMSIZE, T3D, TICKLEN, TITLE, MAX_VALUE, MIN_VALUE, NSUM, POLAR, XLOG, YNOZERO, YLOG, ZLOG

Example

The following example creates an IONPlotApplet that connects to the same server used by the "Connection" applet defined in the IONGraphicApplet example. The applet generates some data for the X value of the plot, and sets the "Title" and "Linestyle" properties of the plot.

```
<APPLET NAME="PLOT" CODE=IONPlotApplet.class
        WIDTH=200 HEIGHT=200>
```

```
    <! This applet uses the applet 'CONNECTION' to connect
        to the server>
    <PARAM NAME="ION_CONNECTION_NAME" VALUE="CONNECTION">
    <PARAM NAME="LINK_URL" VALUE="plotappletsrc.html">
    <PARAM NAME="X_VALUES" VALUE="exp(-(shift(dist(30), 15,
        15)/7)^2)">
    <PARAM NAME="TITLE" VALUE="Plot">
    <PARAM NAME="LINESTYLE" VALUE="2">
</APPLET>
```

IONSurfaceApplet

The IONSurfaceApplet displays an IDL Surface plot. The X, Y and Z values of the plot and any IDL Surface properties supported by ION can be set through parameters to the applet.

Note You can also create surface plots using the IONGraphicApplet, specifying the plot properties in IDL command strings. The IONSurfaceApplet is merely a simplified way to display surface plots.

Parameters

In addition to the parameters described in "Parameters Specified via PARAM Tags" on page 14, the IONSurfaceApplet accepts the following parameters:

X_VALUES

Set the value of this parameter equal to a valid IDL expression that evaluates to a vector or two-dimensional array specifying the X coordinates for the surface. If `X_VALUES` specifies a vector, each element specifies the X coordinate for a column in the `Z_VALUES` array(e.g., X[0] specifies the X coordinate for Z[0,*]). If `X_VALUES` specifies a two-dimensional array, each element specifies the X coordinate of the corresponding point in the `Z_VALUES` array.

Y_VALUES

Set the value of this parameter equal to a valid IDL expression that evaluates to a vector or two-dimensional array specifying the Y coordinates for the surface. If `Y_VALUES` specifies a vector, each element specifies the Y coordinate for a column in the `Z_VALUES` array(e.g., Y[0] specifies the Y coordinate for Z[0,*]). If `Y_VALUES` specifies a two-dimensional array, each element specifies the Y coordinate of the corresponding point in the `Z_VALUES` array.

Z_VALUES

Set the value of this parameter equal to a valid IDL expression that evaluates to a one- or two-dimensional array containing the values that make up the surface. If the `X_VALUES` and `Y_VALUES` parameters are provided, the contour is plotted as a function of the (X, Y) locations specified by their contents. Otherwise, the surface is generated as a function of the two-dimensional array index of each element of `Z_VALUES`.

surface_property_1, ..., surface_property_*n*

Here, `surface_property_*` is the name of a surface property supported by the IONGrSurface class. Properties for the IONSurfaceApplet reflect the capabilities implemented in keywords to the IDL SURFACE procedure.

Note Unlike the IONGraphicApplet `IDL_COMMAND_*` parameter, the `surface_property` parameters are not numbered.

The following IDL Surface properties are supported by the IONSurfaceApplet. Refer to the IDL documentation on keywords available for use with the SURFACE procedure for an explanation of each property:

AX, AZ, BOTTOM, HORIZONTAL, LEGO, LOWER_ONLY, SAVE, SHADES, UPPER_ONLY, ZAXIS, BACKGROUND, CHARSIZE, CLIP, COLOR, DATA, DEVICE, FONT, LINESTYLE, NOCLIP, NODATA, NOERASE, NORMAL, POSITION, SUBTITLE, T3D, TICKLEN, TITLE, MAX_VALUE, MIN_VALUE, NSUM, POLAR, XLOG, YNOZERO, YLOG, XCHARSIZE, YCHARSIZE, ZCHARSIZE, XGRIDSTYLE, YGRIDSTYLE, ZGRIDSTYLE, XMARGIN, YMARGIN, ZMARGIN, XMINOR, YMINOR, ZMINOR, XRANGE, YRANGE, ZRANGE, XSTYLE, YSTYLE, ZSTYLE, XTICKFORMAT, YTICKFORMAT, ZTICKFORMAT, XTICKLEN, YTICKLEN, ZTICKLEN, XTICKNAME, YTICKNAME, ZTICKNAME, XTICKS, YTICKS, ZTICKS, XTICKV, YTICKV, ZTICKV, XTITLE, YTITLE, ZTITLE, ZVALUE, ZLOG

Example

The following example creates an IONSurfaceApplet that connects to the same server used by the "Connection" applet defined in the IONGraphicApplet example. The applet generates some data for the Z value of the plot, and sets the "Title" and "Lego" properties of the plot.

```
<APPLET NAME="SURFACE" CODE=IONSurfaceApplet.class
        WIDTH=200 HEIGHT=200>
    <! This applet uses the applet 'CONNECTION' to connect
        to the server>
    <PARAM NAME="ION_CONNECTION_NAME" VALUE="CONNECTION">
    <PARAM NAME="LINK_URL" VALUE="surfaceappletsrc.html">
    <PARAM NAME="Z_VALUES" VALUE="exp(-(shift(dist(30), 15,
    15)/7)^2)">
    <PARAM NAME="TITLE" VALUE="Surface">
    <PARAM NAME="LEGO" VALUE="1">
</APPLET>
```

ION Applets and Scripting Languages

You can use scripting languages such as JavaScript and VBscript to control ION applets included on an HTML page by calling ION methods that are available to all applets. Communication between scripts and applets gives you a simple way to create interactive HTML pages that build on ION's pre-built applets.

Browser and Script Language Differences

Two competing scripting languages are currently available for use in HTML pages — JavaScript and VBscript. JavaScript was developed by Netscape for use in its Navigator browser; VBscript was developed by Microsoft for use in its Internet Explorer browser. While the two scripting languages have much in common, they do differ in ways that are beyond the scope of this manual to describe. In the context of writing scripts that communicate with ION applets, the important differences are:

- Netscape browsers have a mechanism called "LiveConnect" that allows communication between JavaScripts and applets.
- While Microsoft browsers support JavaScript as well as VBscript, they do not allow communication between JavaScript and applets. In Microsoft browsers, communication between scripts and applets must occur through VBScript.

  The practical result of this situation is that in order to create HTML pages that allow users of both Netscape's Navigator and Microsoft's Internet Explorer to interact with ION applets via scripts, you must write HTML code that decides "on the fly" which scripting language to use.

Choosing Between JavaScript and VBscript

The simplest way to provide pages that use JavaScript for Netscape browsers and pages that use VBScript for Microsoft browsers is to use a "gateway" HTML page that loads one of two other HTML pages depending on the type of browser. The following HTML page uses JavaScript statements to detect whether the browser accessing the page is Netscape Navigator. If so, it loads a JavaScript version of the HTML page; otherwise it loads a VBScript version of the HTML page.

```
<HTML>
<! This page refers IE or Netscape to the proper ION example >
<SCRIPT language=JavaScript>
// <!--
var browser = navigator.appName;

if (browser.indexOf ("Netscape") != -1)
   location = "javascript.html";  // jump to JavaScript page
else
   location = "vbscript.html";    // jump to VBScript page
// -->
</SCRIPT>
</HTML>
```

Note that the script above assumes that the browser is either Navigator or Internet Explorer. Currently, the vast majority of browsers in use are one of these two; still, you may wish to make your own "gateway" HTML page more robust.

Methods Available

The following methods are available for communication between scripting languages and ION applets:

executeIDLCommand('*string*')

where *string* is a valid IDL command string. The `executeIDLCommand()` method allows you to execute any IDL command via a script, with IDL's output going to the specified applet's drawing area.

For example, if you have an IONSurfaceApplet named MYSURF, you could use the following JavaScript statement to change the colortable when the user presses a button:

```
document.MYSURF.executeIDLCommand("LOADCT, 5");
```

See "Example: Using JavaScript" below for a more complete discussion.

disconnect()
Use this method to disconnect from the ION Server.

Example: Using JavaScript

The following HTML code demonstrates the use of JavaScript to interactively update an ION graphic. The example includes an IONGraphicApplet that displays a shaded surface, uses a JavaScript `select` object to create a pulldown list of rotation values, and adds a button to rotate the surface to the selected angle. The line numbers are provided to aid in discussion; they are not part of the HTML code.

Lines 2 - 14 define the HTML header. Note that the JavaScript is included in the HEAD section.

```
1)   <HTML>
2)   <HEAD>
3)    <TITLE>Simple JavaScript Applet Test</TITLE>
```

The script language is JavaScript. We declare the variable rotation with an initial value of 30 degrees.

```
4)    <SCRIPT language=JavaScript>
5)      var rotation = "30";
```

The `getSelectedValue()` function returns the text associated with the value chosen from the pulldown list created in lines 28 - 35.

```
6)    function getSelectedValue(sel) {
7)      return sel.options[sel.selectedIndex].text
8)    }
```

The rot_surf() function retrieves the rotation value and executes the IDL command to redraw the graphic. It is called when the button created in lines 36 - 37 is clicked.

```
9)    function rot_surf() {
10)     rotation = getSelectedValue(document.command_form.rot_value);
11)     document.SURFAPP.executeIDLCommand("SHADE_SURF, a, AZ="+rotation);
12)   }
13)   </SCRIPT>
14)  </HEAD>
```

```
15)  <BODY>
```

JavaScript input controls must be contained in an HTML form.

```
16)  <FORM NAME="command_form">
```

Lines 17 - 26 create an IONGraphicApplet applet named "SURFAPP" that generates some data and creates a shaded surface. Note that the CODEBASE attribute is set to "../classes". This is the proper path for the example as installed with the ION documentation files.

```
17)  <APPLET NAME="SURFAPP" CODE=com.rsi.ion.IONGraphicApplet.class
18)     CODEBASE="../classes" WIDTH=200 HEIGHT=200>
19)     <PARAM NAME="DEBUG_MODE" VALUE="YES">
20)     <PARAM NAME="SERVER_DISCONNECT" VALUE="NO">
21)     <PARAM NAME="DECOMPOSED_COLOR" VALUE="NO">
22)     <PARAM NAME="IDL_COMMAND_0"
23)            VALUE="a = EXP(-(SHIFT(DIST(30), 15, 15)/7)^2)">
24)     <PARAM NAME="IDL_COMMAND_1" VALUE="LOADCT, 5">
25)     <PARAM NAME="IDL_COMMAND_2" VALUE="SHADE_SURF, a">
26)  </APPLET>
27)  <BR>
```

Lines 28 - 35 create the pulldown menu of rotation values.

```
28)  <SELECT NAME="rot_value" SIZE=1>
29)  <OPTION VALUE=15>15
30)  <OPTION VALUE=30 SELECTED>30
31)  <OPTION VALUE=45>45
32)  <OPTION VALUE=60>60
33)  <OPTION VALUE=75>75
34)  <OPTION VALUE=90>90
35)  </SELECT>
```

Lines 36 - 37 create the "Rotate Surface" button, which calls the JavaScript function `rot_surf()`.

```
36)  <INPUT TYPE=BUTTON NAME="rot_button" VALUE="Rotate Surface"
37)  onClick="rot_surf()">
38)  </FORM>
39)  </BODY>
40)  </HTML>
```

See "Notes on the Differences Between the JavaScript and VBScript Versions" on page 30.

Example: Using VBScript

The following HTML code demonstrates the use of VBScript to interactively update an ION graphic. The example includes an IONGraphicApplet that displays a shaded surface, uses a VBScript `select` object to create a pulldown list of rotation values, and adds a button to rotate the surface to the selected angle. The line numbers are provided to aid in discussion; they are not part of the HTML code.

Lines 2 - 12 define the HTML header. Note that the VBScript is included in the HEAD section.

```
1)   <HTML>
2)   <HEAD>
3)    <TITLE>Simple VBScript Applet Test</TITLE>
```

The script language is VBScript. We declare the variable rotation with an initial value of 30 degrees.

```
4)    <SCRIPT language=VBScript>
5)      Dim rotation
6)      rotation = "30"
```

The `rot_button_OnClick()` subroutine retrieves the index of the value selected in the pulldown list created in lines 27 - 34, uses the index to retrieve the text value, and executes the IDL command to redraw the graphic.

```
7)    sub rot_button_OnClick()
8)      ind = document.command_form.rot_value.selectedIndex
9)      rotation = document.command_form.rot_value.options(ind).value
10)     document.SURFAPP.executeIDLCommand("SHADE_SURF, a, AZ="+rotation)
11)   end sub
12)   </SCRIPT>
13)  </HEAD>
14)  <BODY>
```

Lines 15 - 24 create an IONGraphicApplet applet named "SURFAPP" that generates some data and creates a shaded surface. Note that the CODEBASE attribute is set to "../classes". This is the proper path for the example as installed with the ION documentation files.

```
15)  <APPLET NAME="SURFAPP" CODE=com.rsi.ion.IONGraphicApplet.class
16)     CODEBASE="../classes" WIDTH=200 HEIGHT=200>
17)     <PARAM NAME="DEBUG_MODE" VALUE="YES">
18)     <PARAM NAME="SERVER_DISCONNECT" VALUE="NO">
19)     <PARAM NAME="DECOMPOSED_COLOR" VALUE="NO">
```

```
20)     <PARAM NAME="IDL_COMMAND_0"
21)            VALUE="a = EXP(-(SHIFT(DIST(30), 15, 15)/7)^2)">
22)     <PARAM NAME="IDL_COMMAND_1" VALUE="LOADCT, 5">
23)     <PARAM NAME="IDL_COMMAND_2" VALUE="SHADE_SURF, a">
24) </APPLET>
```

VBScript input controls must be contained in an HTML form.

```
25) <FORM NAME="command_form">
26) <BR>
```

Lines 27 - 34 create the pulldown menu of rotation values.

```
27) <SELECT NAME="rot_value" SIZE=1>
28) <OPTION VALUE=15>15
29) <OPTION VALUE=30 SELECTED>30
30) <OPTION VALUE=45>45
31) <OPTION VALUE=60>60
32) <OPTION VALUE=75>75
33) <OPTION VALUE=90>90
34) </SELECT>
```

Line 35 creates the "Rotate Surface" button. The `rot_button_OnClick()` VBScript subroutine is called automatically when this button is clicked.

```
35) <INPUT TYPE=BUTTON NAME="rot_button" VALUE="Rotate Surface">
36) </FORM>
37) </BODY>
38) </HTML>
```

Notes on the Differences Between the JavaScript and VBScript Versions

1. Interaction between the applet and the script language takes place in JavaScript statements in the Netscape Navigator version, and in VBScript statements in the Microsoft Internet Explorer version. The syntax of the scripting language is slightly different.
2. In the JavaScript version, the applet is included within the HTML FORM definition. Internet Explorer requires that the applet be located outside the FORM.
3. In JavaScript, you must explicitly tie a control (a button, for example) to a JavaScript function. VBScript automatically looks for a subroutine name based on the name of the button.

Chapter 4

ION Java Classes

The following topics are covered in this chapter:

What are the ION Graphics Classes?

The ION Graphic Component Java Classes are a set of high level Java classes that provide a rapid and powerful way to include IDL graphics in a Java application or Java applet. The Graphic Component classes are built from the ION Low-Level classes, and provide a simpler interface, which allows you to connect to the ION Server and display graphics generated by IDL. The ION Graphics classes were developed using the Java Developer's Kit (JDK) version 1.02.

IONGr Graphic Objects

IONGr Graphic objects are high-level objects that encapsulate ION connections and various types of graphics.

IONGrConnection

An IONGrConnection object provides a connection between the ION Server and the client. In addition to establishing and ending the connection, IONGrConnection allows you to get and set the values of IDL variables on the ION Server, add and remove drawable objects to the connection, and execute IDL commands directly.

IONGrContour

An IONGrContour object represents a contour graphic. IONGrContour allows you to get and set properties of the contour plot (via keywords to the IDL CONTOUR routine) and to draw the contour object. IONGrContour extends IONGrGraphic.

IONGrDrawable

An IONGrDrawable object creates a drawing area that presents graphics produced by the ION Server. IONGrDrawable allows you to configure the drawing area to draw one or more objects, add and remove graphic objects from a drawable, and execute IDL commands directly. Objects of this type can be inserted into the AWT tree.

IONGrGraphic

An IONGrGraphic object provides methods used to manage graphic properties. The other IONGr objects implement this object. IONGrGraphic allows you to get and set graphic properties, and to manage property lists for the graphic object.

IONGrPlot

An IONGrPlot object represents an plot graphic. IONGrPlot allows you to get and set properties of the plot (via keywords to the IDL PLOT routine) and to draw the plot object. IONGrPlot extends IONGrGraphic.

IONGrSurface

An IONGrSurface object represents a surface graphic. IONGrSurface allows you to get and set properties of the surface (via keywords to the IDL SURFACE routine) and to draw the surface object. IONGrSurface extends IONGrGraphic.

ION Graphic Objects

ION Graphic objects extend the IONGrDrawable object and contain one of the ION graphic type objects (plot, surface, contour). ION Graphic objects can be inserted into a Java AWT tree.

IONContour

An IONContour object represents a contour graphic and a drawing area. IONContour allows you to get and set properties of the contour (via keywords to the IDL CONTOUR routine) and to draw the contour object. IONContour extends IONGrDrawable and includes an IONGrPlot object.

IONPlot

An IONPlot object represents a plot and a drawing area. IONPlot allows you to get and set properties of the plot (via keywords to the IDL PLOT routine) and to draw the pot object. IONPlot extends IONGrDrawable and includes an IONGrPlot object.

IONSurface

An IONSurface object represents a surface graphic and a drawing area. IONSurface allows you to get and set properties of the surface (via keywords to the IDL SURFACE routine) and to draw the surface object. IONSurface extends IONGrDrawable and includes an IONGrSurface object.

Using the Graphics Classes

The ION Graphics classes have a number of common features. The contour, plot, and surface objects all allow you to set the data values, retrieve and set properties, and draw the object. See the reference information on each object for a complete list of methods.

Setting Values

The ION Graphics objects that include data all allow you to set the initial data values when you create the object. You can also reset the data values using the `setXValue` / `setYValue` / `setZValue` methods. The `set` methods enable you to change the value of the displayed data on the fly without re-creating the object in question.

Getting and Setting Properties

The contour, plot, and surface objects can all be modified by changing the value of a set of properties associated with the objects. The list of properties available for modification is a subset of the list of properties controlled by keywords to the corresponding IDL Direct Graphics routine (CONTOUR, PLOT, or SURFACE). Consult the IDL Reference Guide for details about the settings for individual properties.

Drawing

With the exception of the IONGrConnection object, all of the ION Graphics objects have a draw method. Calling the `draw` method on a given object causes it to be displayed in the associated drawing area.

What are the Low-Level Classes?

The ION Low-Level classes are the most basic building blocks of ION applications. They provide a degree of control not available to the user of the ION Graphics Component classes; therefore they require more sophisticated Java programming skills to use. The ION Low-Level classes were developed using the Java Developer's Kit (JDK) version 1.02.

IONCallableClient

IONCallableClient provides mechanisms to handle communication with the server, execution of IDL commands, retrieving IDL command log output and the getting and setting of IDL variables on the ION Server. Objects of this type can be inserted into the AWT tree.

IONCanvas

This class represents a visible drawing area upon which graphics can be displayed.

IONCommandDoneListener

This interface defines the method an object must implement to receive notification that an IDL command has completed.

IONComplex

This class represents a single-precision complex number.

IONDComplex

This class represents a double-precision complex number.

IONDrawable

This interface defines the methods that an object must implement to act as an ION drawable object. An ION drawable is an object that can be drawn to by an IONGraphicsClient.

IONGraphicsClient

This class provides mechanisms to handle the processing of graphic primitive data sent from the ION Server. Information sent by the server is read by mechanisms provided by the superclass IONCallableClient and dispatched to the `handleServerAction()` method of this class.

IONMouseListener

This interface defines the callback methods that an object must define to be notified of mouse events occurring on an object that implements the IONDrawable interface.

IONOffScreen

This object represents an invisible drawing area on which graphic output can be placed.

IONOutputListener

This interface defines the method that an object must implement to receive ION Server output text.

IONPaletteFilter

This class is used to implement an ION version of a Java ImageFilter. This image filter is used to change the color pallet (color table) being used. The Java Image scheme uses an image filter object to change attributes and perform operations on images; objects of the IONPalletFilter class are used to change the color table when using an indexed-color model system.

IONVariable

This object is a client side representation of an IDL variable. IONVariable objects are used to read and write data between the IDL server and clients.

IONWindow

This class extends the Java Frame class to let ION windows have access to top-level window events. Objects of the IONWindow class need an object of the IONWindowingClient class registered with them as a callback object in order to handle "destroy" events.

IONWindowingClient

This class provides mechanisms to handle the processing of the windowing commands that are part of an IDL Direct graphics driver. This includes the creation, deletion, showing, hiding, and iconization of windows on the client.

Chapter 5

Building ION Applets and Applications

The following topics are covered in this chapter:

It is beyond the scope of this manual to discuss all of the elements that go into creating a Java applet. We assume that you are already familiar with the process of creating applets, and focus here on details that are specific to building ION applets.

You will find details on the ION Java classes used to build ION applets in "ION Java Classes" on page 31 and in the "ION Class and Method Reference" on page 69.

Creating Applets

When creating your ION applet, keep the following points in mind.

Import the ION Package

In addition to the standard Java packages (and any other packages used in your applet), you must import the ION package with the statement:

```
import com.rsi.ion.*
```

ION Applets Extend the Java Applet Class

ION applets are subclassed from (they *extend*) the Java Applet class. When defining your applet class, use a statement like the following:

```
public class MyIONApplet extends Applet
```

where *MyIONApplet* is the name of your applet class.

See "Simple Applet Example" on page 42 for an example.

Compiling Applets

Keep the following points in mind when you compile the `.java` file that contains your applet code into an applet:

Set the Class Path

When you compile your applet, the ION class files must be in the Java complier's class path. Depending on your specific Java compiler, you may set the class path by defining the CLASSPATH environment variable or by changing settings in a dialog.

Since ION is a *package*, the class files are stored in a directory structure. The name of the ION package is `com.rsi.ion`, so the classes are located in the following directory:

ROOT_DIR`/com/rsi/ion`

Where *ROOT_DIR* is the path to the `classes` subdirectory of the main ION directory. For example, if you have installed ION in the directory `/usr/local/rsi/ion`, the actual location of the ION class files would be:

```
/usr/local/rsi/ion/classes/com/rsi/ion
```

and *ROOT_DIR* would be

```
/usr/local/rsi/ion/classes
```

In this case, you would set your CLASSPATH environment variable with the following shell command:

```
setenv CLASSPATH ".;/usr/local/rsi/ion/classes"
```

(or however you set env variables on your system). The Java compiler will add the `com/rsi/ion` portion of the path when it looks for the package.

Once the CLASSPATH is set, you can compile your code with a shell command like the following:

```
javac myIONApplet.java
```

where *`myIONApplet`* is the name of your applet.

Including Applets in HTML Pages

To include your compiled applet in an HTML page, use the `<APPLET>` tag with the `NAME`, `CODE`, `WIDTH`, and `HEIGHT` attributes:

```
<APPLET NAME="myIONApplet" CODE=myIONApplet.class
             WIDTH=300 HEIGHT=300 >
</APPLET>
```

For more information, see "Using ION Applets" on page 12.

Locating the Class Files for use by ION Applets

ION applets must have access to the ION class files in order to run. While you can use the CODEBASE attribute to specify a relative path from the location of an HTML page containing an ION applet tag to the location of the class files, it is often easier to copy the class files (or provide a symbolic link, if your system supports symbolic links) to another directory located in or near the directory containing your HTML files.

For example, suppose you have located your HTML pages in a directory named `public_html`. You may wish to place the ION package, the ION zip file, and the ION jar file in a subdirectory of `public_html` named `java`. If you then include any ION applet class files you create in the `java` directory, you could simply specify:

```
CODEBASE="./java"
```

in the `<APPLET>` tag used in your HTML page.

See the discussion of the CODEBASE attribute on page 13 for further details.

Supporting Java Archive Files

When a web browser encounters an HTML page that contains a Java applet, the class files that make up the applet are downloaded from the web server into the browser. The applet is executed only after all of the necessary class files have been downloaded. Because a separate HTTP connection between the client and the server is established for each class

file, the download time for a large applet (an applet with many class files) can be substantial.

To increase the download performance of Java applets, the Java 1.1 specification includes the concept of a *Java ARchive* file, or `jar` file, that can contain multiple class files, thus avoiding the need for multiple connections. A `jar` file can also be compressed, further speeding the download process. Unfortunately, many of the web browsers in current use do not support the `jar` format. To make matters even more complicated, some browsers that do not support the compressed `jar` format do support uncompressed archives in `zip` format.

To support the different methods used by different browsers to download Java class files, ION provides three separate versions of the ION class library. These are:

1. The raw Java class files, contained in the `com/rsi/ion` directory structure installed in the `classes` directory of the ION distribution. Each file is downloaded to the browser via a separate connection to the server. The raw Java class files are used by browsers that don't support the ARCHIVE attribute to the APPLET tag. For example, version 3 of Microsoft's Internet Explorer does not support the ARCHIVE attribute.

2. An uncompressed `zip` file named `ion_`*`release`*`.zip` that contains all of the Java class files included in the ION package, *with the exception of the class files for the ION pre-built applets*. This `zip` file is located in the `classes` directory of the ION distribution, and can be downloaded via a single connection to the server. The `zip` file can be used by browsers that support the ARCHIVE attribute, but which don't support compressed archive files. For example, version 3 of Netscape's Navigator supports the ARCHIVE attribute but does not support `jar` files.

3. A compressed Java ARchive (`jar`) file named `ion_`*`release`*`.jar` that contains the Java class files included in the ION package, *with the exception of the class files for the ION pre-built applets*. This jar file is located in the `classes` directory of the ION distribution, and can be downloaded via a single connection to the server. The `jar` file can be used by browsers that support compressed archive files. For example, version 4 and later of Netscape's Navigator supports `jar` files.

Supporting Multiple Browser Types

Use the following procedure to create a set of HTML pages that will use the most efficient download method for any of the three browser types defined above.

1. Ensure that the archive files and the root of the unarchived package hierarchy are all in the same directory. By default, all three are located in the `classes` subdirectory of the ION distribution. This directory should be specified via the CODEBASE attribute to the APPLET tag.

2. Create two versions of each HTML page that contains an ION applet. One page should include a reference to the uncompressed archive file via the ARCHIVE attribute to the APPLET tag (`ARCHIVE=ion_`*`version`*`.zip`). The other page should include a reference to the compressed archive file (`ARCHIVE=ion_`*`version`*`.jar`). Browsers that do not support the ARCHIVE attribute will ignore it and download the unarchived files.

3. Create a "switch page" that includes JavaScript. The switch page determines which version of the browser is present and loads the appropriate HTML page.

```
<SCRIPT language="JavaScript">
<!--
   navigator.onerror = null;
   version = (  parseInt(navigator.appVersion) > 3 ? "4" : "3");
   if(version == "4"){
   // Version 4 can handle jar files, load the Jar page
      location.replace("<JAR_page>.html");
   }else{
      location.replace("<ZIP_page>.html");
}
// -->
</SCRIPT>
```

where *<JAR_PAGE>*`.html` is the name of the HTML page that references the `ion_`*release*`.jar` file and *<ZIP_PAGE>*`.html` is the name of the HTML page that references the `ion_`*release*`.zip` file. For example, you may name the page that references the JAR file `myfile_j.html` and the file that references the ZIP file `myfile_z.html`.

Error Handling and ION Exceptions

When the ION Server detects an error, it returns an exception value you can detect and act upon using error-handling code. Consult the reference page for the method you are using to determine which exceptions ION can detect in a given situation.

Error handling is generally accomplished via a Java `try/catch` code segment. The following skeleton `try/catch` code illustrates how to catch errors and display an error message on the Java console. For a more detailed example, see "Java Applet Examples" on page 27.

Note If an ION method (or any Java method, for that matter) returns a checked exception value, you must handle the exception in your code. The Java compiler will complain if you do not properly handle all possible exception values.

```
try{
   some ION command
}catch(IOException e) {
   // IO Error
      System.err.println("Error: Communication error");
      return;
}catch( IONIllegalCommandException e){
```

```
      // Illegal Command
         System.err.println("Error: Illegal Command");
         return;
   }catch( IONSecurityException e){
      // Security Violation
         System.err.println("Error: Security Violation");
         return;
      }
   }
```

Simple Applet Example

The following Java code creates a simple applet that displays an IDL graphic. The numbers to the left of the code are included for purposes of discussion only; do not include the numbers in your `.java` file. The example constructs an applet named IONGrEx1App; the code is saved in a file named `iongrex1app.java`.

Note The characters "//" denote comments in Java code.

Lines 1-4 import the Java packages used by the applet.

```
1)  import java.awt.*;
2)  import java.applet.*;
3)  import java.io.*;
4)  import java.net.*;
```

Import the ION package.

```
5)  import com.rsi.ion.*;
```

Declare the applet class. This class, IONGrEx1App, is an extension (a subclass of) Java's Applet class.

```
6)  public class IONGrEx1App extends Applet
7)  {
```

Declare the ION Variables that the applet will use. The `c_ionCon` variable will hold the IONGrConnection object, which manages the connection to the ION Server, and the `c_ionDrw` will hold the IONGrDrawable object.

```
8)  IONGrConnection c_ionCon;
9)  IONGrDrawable   c_ionDrw;
```

The init method is the first method called with the applet starts.

```
10) // ****************************
```

```
11)  // Init Method
12)  // ****************************
13)  public void init()
14)    {
```

Create ION objects using variables defined in lines 8-9.

```
15)  // Create connection and drawable objects
16)  c_ionCon = new IONGrConnection();
17)  c_ionDrw = new IONGrDrawable(this.size().width
        this.size().height);
```

Line 18 initializes the widget geometry manager, and line 19 adds the ION drawable to the Applet's AWT (widget) tree.

```
18)  setLayout(new GridLayout(1, 1)); // this is the widget geometry man-
        ager.
19)  add(c_ionDrw);
```

Connect to the ION Server. Since the connect method can throw exceptions, the connect method must be wrapped in a try/catch block. Since we are building an applet, which can only connect to the machine it was downloaded from, we first get the host information from the applet class using the `getCodeBase().getHost()`; methods.

```
20)  try {
21)      c_ionCon.connect(this.getCodeBase().getHost());
22)      }catch(IOException e) {
23)         // IO Error
24)            System.err.println("Error: Communication error");
25)            return;
26)      }catch( IONIllegalCommandException e){
27)         // Illegal Command
28)            System.err.println("Error: Illegal Command");
29)            return;
30)      }catch( IONSecurityException e){
31)         // Security Violation
32)            System.err.println("Error: Security Violation");
33)            return;
34)      }
```

Set our color mode so that 8 bit color will work on 8-bit and 24-bit displays.

```
35)  c_ionCon.setDecomposed(false);
```

Add the IONGrDrawable to the ION connection:

```
36)  c_ionCon.addDrawable(c_ionDrw);
```

Send the IDL commands to the server. Must wrap in a try/catch block just in case there is an IO/Network error.

```
37)  // Issue IDL commands to generate a plot
38)      try {
39)        // Set the color table
40)        c_ionCon.executeIDLCommand("loadct, 15");
41)  // Create some data
42)        c_ionCon.executeIDLCommand("a = dist(30)");
43)  // Draw a contour plot
44)        c_ionCon.executeIDLCommand("show3, a");
45)        } catch(IOException e) {
46)        System.err.println("Error: Communication error");
47)      // Write the message on the applet
48)         Graphics g = c_ionDrw.getGraphics();
49)         g.setColor(Color.red);
50)       g.drawString("Error: Communication error", 5, size().height/2);
51)      }
52)    }
```

The destroy method is called when the browser kills the applet. We override it here so that we can disconnect from the ION Server. This is very important — if the user doesn't disconnect from the server when the applet is destroyed, the browser may stop responding until the connection times out (about a minute).

```
53)  public void destroy(){c_ionCon.disconnect();}
54)  }
```

Further Examples

Example code illustrating ION features is included in the installed ION distribution. You will find example HTML files located in subdirectories of the examples directory in your installed ION distribution. Browse through the example code starting with the file `examples.html` in the `examples` directory. Many of the examples allow you to view the Java source for the example within your browser. The HTML version of the *ION User's Guide* (this document) includes links to the installed example HTML files.

The raw Java source files for the example ION classes are included in the `src` subdirectory of the `classes` directory. Also included in the src subdirectory are a number of IDL .pro files that are called by the ION demonstration applets. In order to use the ION demos on

your own system, you must add the directory *RSI_DIR*/ion_1.1/classes/src to IDL's path. See "Location of IDL .pro Files" on page 223.

Note For the examples to function properly, you must have the ION Server running on your machine. If you do not yet have the ION Server running on your system, visit Research Systems' ION web site and view example code there.

Debug Mode

The IONGrConnection object supplies a debugMode() method that allows you to view the IDL command log output by holding down the Shift key and clicking on the ION drawing area associated with the connection. Set debug mode by adding the following to the Java code that establishes the connection to the ION Server:

```
connection.debugMode(true);
```

where *connection* is the IONGrConnection object.

When debug mode is in effect, holding down the Shift key and clicking on the ION drawing area will pop up a separate window that displays the output that would appear in the IDL command log.

The ION Device

IDL uses the concept of a *current graphics device* when creating and displaying IDL Direct Graphics. When the ION Server requests graphics from IDL, it automatically sets the current graphics device to `'ion'`; graphics output from IDL is sent directly to the ION Server. You do not need to explicitly set the graphics device to `'ion'` unless you have explicitly used the IDL SET_PLOT procedure to change the current device to some other device.

For example, suppose you wish to include a "Print" button in a Java applet. Your applet might include something like the following:

```
executeIDLCommand("SET_PLOT, 'printer'")
execute more IDL commands to draw an image on the printer
executeIDLCommand("SET_PLOT, 'ion'")
```

Keywords Accepted by the ION Device

The following keywords to the IDL DEVICE routine are available when the current graphics device is set to `'ion'`. Except where indicated, keywords to the ION device work just as they do for other IDL graphics devices.

COPY

Use this keyword to copy a rectangular area of pixels from one region of a window to another. COPY should be set a six or seven element array: $[X_s, Y_s, N_x, N_y, X_d, Y_d, W]$, where: (X_s, Y_s) is the lower left corner of the source rectangle, (N_x, N_y) are the number of

columns and rows in the rectangle, and (X_d, Y_d) is the coordinate of the destination rectangle. Optionally, *W* is the index of the window *from which the pixels should be copied* to the *current* window. If it is not supplied, the current window is used as both the source and destination.

DECOMPOSED

This keyword is used to control the way in which graphics color index values are interpreted when using displays with decomposed color (TrueColor visuals). This keyword has no effect with other types of visuals.

Set this keyword to 1 to cause color indices to be interpreted as 3, 8-bit color indices where the least-significant 8 bits contain the red value, the next 8 bits contain the green value, and the most-significant 8 bits contain the blue value. This is the way IDL has always interpreted pixels when using visual classes with decomposed color.

Set this keyword to 0 to cause the least-significant 8 bits of the color index value to be interpreted as a PseudoColor index. This setting allows users with TrueColor displays to use IDL programs written for standard, PseudoColor displays without modification.

In older versions of IDL, color index values higher than !D.N_COLORS-1 were clipped to !D.N_COLORS-1 in the higher level graphics routines. In some cases, this clipping caused the exclusive-OR graphics mode to malfunction with raster displays. This clipping has been removed. Programs that incorrectly specified color indices higher than !D.N_COLORS-1 will now probably exhibit different behavior.

FONT

Set this keyword to a scalar string specifying the name of the font used when the hardware font is selected.

Note The hardware fonts available are supplied by Java itself, not the platform on which IDL is running. Java's font system supplies several standard fonts, with the following font names: "Helvetica", "TimesRoman", "Courier", "Dialog", "DialogInput", "ZapfDinbats", and "default". These font names will map to different actual fonts on different platforms, but will always be handled gracefully by Java. If you specify a different font, Java will substitute one of the standard fonts automatically.

Note that hardware fonts cannot be rotated, scaled, or projected, and that the "!" commands accepted for vector fonts for subscripts and superscripts may not work. When generating three-dimensional plots, it is best to use the vector-drawn characters because IDL can draw them in perspective with the rest of the plot.

The GET_FONTNAMES keyword, described below, can be used to retrieve a list of available fonts.

The FONT keyword should be set to a string with the following form:

DEVICE, FONT="*font**$modifier_1$*$modifier_2$*...$modifier_n$"

where the asterisk (*) acts as a delimiter between the font's name (*font*) and any modifiers. The string is *not* case sensitive. Modifiers are simply "keywords" that change aspects of the selected font. Valid modifiers are:

- For font size: Any number is interpreted as the point size of the font to use.
- For font weight: PLAIN, BOLD
- For font angle: ITALIC

For example, the following commands tell ION to use hardware fonts, change the font, and then make a simple plot:

```
executeIDLCommand("!P.FONT = 0")
executeIDLCommand("DEVICE, FONT = 'HELVETICA*ITALIC*24'")
executeIDLCommand("PLOT, FINDGEN(10), TITLE = 'IDL Plot'")
```

GET_CURRENT_FONT

Set this keyword to a named variable in which the name of the current font is returned as a scalar string.

GET_FONTNAMES

Set this keyword to a named variable in which a string array containing the names of available fonts is returned. If no fonts are found, a null scalar string is returned. This keyword must be used in conjunction with the FONT keyword. Set the FONT keyword to a scalar string containing the name of the desired font or a wildcard.

GET_GRAPHICS_FUNCTION

Set this keyword to a named variable that returns the value of the current graphics function (which is set with the SET_GRAPHICS_FUNCTION keyword). This can be used to remember the current graphics function, change it temporarily, and then restore it. See the SET_GRAPHICS_FUNCTION keyword for an example.

GET_SCREEN_SIZE

Set this keyword to a named variable in which to return a two-word array that contains the width and height of the server's screen, in pixels.

SET_CHARACTER_SIZE

The standard size and vertical spacing of vector-drawn fonts can be changed by specifying this keyword with a two-element vector. The first element specifies the new character width and thus the height of the characters (because vector-drawn fonts have a fixed aspect ratio). The second element specifies the vertical distance between lines. The default produces a character that is approximately 8 pixels wide, with 12 pixels between lines.

SET_GRAPHICS_FUNCTION

Most window systems allow applications to specify the graphics function. This is a logical function which specifies how the source pixel values generated by a graphics operation are combined with the pixel values already present on the screen. ION supports only the following two of the fifteen graphics functions supported by IDL Direct Graphics:

Logical Function	Code	Definition
GXcopy	3	source

Logical Function	Code	Definition
GXxor	6	source XOR destination

The default graphics function is GXcopy, which causes new pixels to completely overwrite any previous pixels. Not all functions are available on all window systems.

See "IDL Graphics Devices" in the IDL Reference Guide for more information about how IDL handles graphics devices. Note that because ION was released after IDL version 5.0, the ION device does not appear in the list of supported devices in the IDL 5.0 documentation.

Tips and Tricks

This section includes suggestions that may be useful in some situations. Make sure your installation meets the criteria defined here before implementing any of these suggestions.

Local Netscape Users

If your installation is used only by a known set of users who all use Netscape's Navigator version 4 or later, you can eliminate the need to download the Java class files when an applet loads. Do the following:

1. Have each of your users install a copy of the `ion_release.jar` file in the `Program/java/classes` subdirectory of their local Netscape directory.
2. Remove the ARCHIVE attribute from the APPLET tag in your HTML code.

Caution The Java security mechanism requires that applet classes must be loaded from the server on which ION is running. This means that the approach described here will fail with a security error if the applet class files are not located in the `com/rsi/ion` subdirectory of the directory specified by the CODEBASE attribute.

Destroy Methods

If your applet includes a `destroy()` method, it will be invoked automatically when the browser shuts down. Depending on the browser, the `destroy()` method may also be invoked when a user resizes page containing the applet, opens a new web page, or performs some other action.

It is good practice to include a `destroy()` method in your applets that closes the ION connection and does any other cleanup that may be necessary.

Client-side Animation

IDL's animation routines all rely on the IDL widget toolkit, and are thus not suitable for use with ION. You can, however, use IDL to create the individual frames of an animation and create an ION applet to build an array of frames and display the animation on the client side (in a browser or Java application).

An example application that does this sort of client-side animation is included in the ION distribution. Point your browser at the file `animation.html` in the `demo` subdirectory of the `examples` directory. The Java sources for the animation classes are included in the `src` subdirectory of the `classes` directory. Note that the animation demo relies on an IDL `.pro` file; see "About the Example Code" on page 5.

Deciding Which Canvas Graphics Object to Use

There are two distinct graphics canvases available to you as a creator of ION applications. One canvas is created and managed by IDL, the other by the Java graphics system. In the event that you wish to draw directly onto the displayed image (when creating an annotation or selecting a region of interest, for example), you will have to decide which of the two canvases is appropriate.

If you decide to draw directly on the ION canvas, use the `getIONGraphics()` method on an IONDrawable object to get access to the drawable area created by IDL. If you prefer to do your drawing independent of the IDL graphics, use the `getGraphics()` method instead.

An example application that draws a region of interest bounding box on a Java graphics canvas is included in the ION distribution. Point your browser at the file `imageproc.html` in the `demo` subdirectory of the `examples` directory. The Java sources for the image processing and ROI classes are included in the `src` subdirectory of the `classes` directory. Note that the image processing demo relies on an IDL `.pro` file; see "About the Example Code" on page 5.

Chapter 6

Configuring the ION Server

The following topics are covered in this chapter:

ION is a Callable IDL application that processes requests from clients and returns graphics and data to the client. ION consists of two applications: the *ION Daemon*, which watches a port on the server machine and sets up a connection between the client application (a Java applet) and the *ION Server process*. The ION Server process accepts requests from the client application, uses IDL to manipulate data and generate graphics, and returns the results to the client.

Normally, ION uses a persistent two-way socket connection between the client and the server. If your site is isolated from the bulk of the Internet behind a *firewall*, persistent two-way connections may not be allowed. ION provides an HTTP *Tunnel Broker* to allow ION sessions to operate through most firewalls. See "The ION HTTP Tunnel Broker" on page 58 for details on the ION Tunnel Broker.

Note ION Servers are available only for systems running one of the following operating systems:

- Digital Unix 4.0 and later
- HP/UX 10.01 and later
- AIX 4.1 and later
- Linux kernel 2.0.18 and later
- Irix 5.3, 6.2 and later
- SunOS 4.1.3 and later
- Solaris 2.5 and later
- Microsoft Windows NT 4.0 and later

Command Security

The ION Server implements a security system based on IDL command filtering. The security system has two internal command lists: one list consists of commands that *are not* allowed to be run on the IDL server process; the other list specifies commands which are allowed. (If an IDL command is included in both lists, it will *not* be allowed to run.)

When an ION client sends an IDL command to the ION Server for execution, the command line is scanned for function and procedure names. These names are first checked against the command inclusion list (commands that can be run on the server), and if the command is not in the list it is rejected. If the command inclusion check passes, the routine is the checked against the command exclusion list (routines that should not be run on the server). If the command is in the command exclusion list it is rejected. If the command passes the exclusion list check, it is sent to the ION Server process for execution.

Note ION's command security configurations are designed to prevent IDL commands from being used in an unauthorized or hostile manner during connections to your ION Server. Remember that you must also properly configure your Web server to prevent unauthorized access to your site via other mechanisms.

Security Command Files

The ION Server allows the user to specify IDL commands to be included or excluded from the server via text files. Inclusion and exclusion text files consist of a single command on each line. Lines that are blank or start with the "#" character are ignored. For example, you could create an ION exclude file containing the following lines:

```
# My commands to prevent
myPlot
myProcess
myFunction
```

Security command files are specified on the command line when the ION Daemon is started. See “The ION Daemon” on page 53 for details.

Client Verification

When the ION Daemon detects an incoming server connection, the daemon verifies that the client is a valid ION client. ION clients are valid if they have been created using the ION Java classes described in this document. If the client is not valid, the daemon rejects the connection and no ION Server process is started.

Connection Limit

There are two limits set on the number of connections the ION Server will accept. If you have specified a maximum number of connections via the `maxconn` switch to the ION Daemon process, the ION Daemon will reject new clients after reaching that limit. If no maximum number of connections is specified to the daemon, the maximum number of connections allowed is defined by the ION Server license. If the limit is reached, the ION Daemon will notify new ION clients that the limit has been reached and will close the connection.

Starting an ION Server Process

Once the client has been verified by the daemon and all other checks have passed, the ION Daemon begins an ION Server process and connects the ION client with the process. Once the client has been connected to the server process, the ION Daemon returns to processing incoming requests. When the ION Server process dies, the daemon is notified.

The ION Daemon

The ION Daemon is a process that listens to a specified socket port, waiting for a communication request. Once a connection is received and verified, the daemon starts up an ION Server process, connects the client to the server process and waits for further connection requests.

Start the ION Daemon process by executing the `iond` command at the shell prompt (under Windows NT, you must open a Command Prompt window to execute the command):

```
$ RSI_DIR/ionrelease/bin/iond [switches]
```

where *RSI_DIR* is the path to the installation directory, `ion`*`release`* is the name of the ION directory, and *[switches]* are optional command-line parameters discussed below.

Note If you have installed and started the ION Daemon service on your Windows NT machine, you do not need to execute the `iond` command manually. See "Installing the ION Daemon Service under Windows NT" on page 61 for details. If you run the `iond` command manually and have specified a log file in the ION Properties dialog (see "ION Windows NT Utilities" on page 62), all ION Daemon output will be redirected to the log file. While the Daemon is functioning normally, *appears* to have "hung," because no output appears on your screen.

The ION Daemon is responsible for the following:

- Parsing command line parameters,
- Establishing the security level and initializing security levels,
- Maintaining server logs,
- Managing the number of current connections,
- Receiving connections and starting ION Server processes,
- Verifying incoming requests as valid ION clients.

Command Line Parameters

The following command line parameters are accepted by the ION Daemon

-exfile=filename
Set this switch to the name of a file that contains a list of IDL commands (procedure or function names) that the server should not accept. Any command that attempts to execute one of the listed routines will be rejected. The file should contain one routine name on each line. Blank lines and lines that begin with the "#" character are ignored.

Specifying an exclude file will not alter the list of routines rejected as a result of the setting of the `security` switch.

-infile=filename
Set this switch to the name of a file that contains a list of IDL commands (procedure or function names) that the server should accept. Any command that attempts to execute a routine that is not in the list will be rejected. The file should contain one routine name on each line. Blank lines and lines that begin with the "#" character are ignored.

Specifying an include file will not alter the list of routines rejected as a result of the setting of the `security` switch.

Note If a routine is *excluded* (either via an exclude file, a list of excluded routines, or via the security switch), it will be rejected even if that routine is also included in an include file or list.

-excomm="routine 0, routine 1,...routine n"
Set this switch to a comma-separated list of IDL commands (procedure or function names) to add to the exclusion list. This switch works in the same way as the `exfile` switch; it is provided as a convenience.

Specifying a list of routines to exclude will not alter the list of routines rejected as a result of the setting of the `security` switch.

-incomm="routine 0, routine 1, ...routine n"
Set this switch to a comma-separated list of IDL commands (procedure or function names) to add to the inclusion list. This switch works in the same way as the `infile` switch; it is provided as a convenience.

Specifying a list of routines to include will not alter the list of routines rejected as a result of the setting of the `security` switch.

Note If a routine is *excluded* (either via an exclude file, a list of excluded routines, or via the security switch), it will be rejected even if that routine is also included in an include file or list.

-idldir=path [Windows NT only]
Set this switch to override the value contained in the Windows registry (or in an the IDL_DIR environment variable) and specify the location of the IDL directory. See "Configuration Details" on page 66 for details.

-iondir=directory path
Set this switch to the location of the ION directory. ION is always located in a directory named `ion` in the directory specified at installation. By default, the ION directory is `/usr/local/rsi/ion`.

-http
Set this switch to start the ION HTTP Tunnel Broker when starting the ION Daemon. See "The ION HTTP Tunnel Broker" on page 58 for details on the ION Tunnel Broker.

-httplog=filename [Windows NT only]
Set this switch to the name of the file in which you wish to save informational messages from the ION Tunnel Broker. If no logfile is specified, messages will be written to the standard output. (Under Unix, you can create a log file by redirecting the output from `ion_httpd` to a log file of your choosing using the normal system output redirection mechanism.)

-httpport
Set this switch to the port number that the ION HTTP Tunnel Broker should watch for connection requests. If you do not specify a value for the `httpport` switch, the ION Tunnel Broker watches port 9085.

-httptimeout=*n*
Set this switch to the number of minutes the ION Tunnel Broker HTTP peer should stay alive for without hearing from the client. A timeout is necessary to close Tunnel Broker peer processes that may be left running if a browser crashes or experiences some other error that disconnects the browser without shutting down the peer process. If *n* is 0 (zero) the peer will never time out.

-logfile=filename [Windows NT only]
Set this switch to the name of the file in which you wish to save informational messages from the ION Daemon. If no logfile is specified, messages will be written to the standard output. (Under Unix, you can create a log file by redirecting the output from `iond` to a log file of your choosing using the normal system output redirection mechanism.)

-maxconn=N
Set this switch to the maximum number of connections that can be active at once. If you do not specify a value for the `maxconn` switch, the maximum number of connections will be equal to the number of IDL licenses you have available.

-port=N
Set this switch to the port number that the ION Daemon should watch for connection requests. If you do not specify a value for the `port` switch, the ION Daemon watches port 7085.

-rutil
Set this switch to allow the utility routines `iondown` and `ionstat` to be run from any host. By default, connections from these routines are allowed only if the routines are run on the same host as the ION Daemon.

-timeout=seconds
Set this switch to the number of seconds ION will wait to receive a response. If no response is received within the timeout interval, ION will make a second attempt (it will "ping" the remote machine). If no response is received within the second timeout interval, ION will close the connection.

The default timeout value is 60 seconds. You may wish to increase the timeout value with extremely slow network connections.

-security=[none, widgets, linking, filein, fileout, fileio, os, device, df]
Set this switch to a comma-separated list of tokens that define a list of IDL routines. IDL routines specified via a token in the security list will not be passed through to the IDL session by the ION Server.

Note Do not include the brackets (`[]`) when specifying the `security` flag.

If you do not include the `security` flag when starting the ION Daemon, the following default tokens are set:

```
widgets, fileio, os, linking, device, df
```

If you include the security flag when starting the ION Daemon, only the tokens you specify are set. See the discussion of the `infile`, `exfile`, `incomm`, and `excomm` flags for further information on specifying which IDL commands will be accepted by daemon.

The `security` switch accepts the following tokens, shown in **bold**. (In the lists below, the asterisk is used to represent all IDL routines of a given type. For example, `WIDGET_*` represents `WIDGET_BASE`, `WIDGET_BUTTON`, `WIDGET_CONTROL`, etc.)

none
No security checking is provided.

device
Disables changing devices using the `SET_PLOT` routine.

widgets
Disables the use of widgets, compound widgets, and widget-based IDL programs by disallowing use of the following routines:

```
ANNOTATE, CW_*, DIALOG_*, EFONT, INSIGHT, MP_WIDGETS,
OS_PICKFILE, PICKFILE, PWIDGET, SLICER, SLIDE_IMAGE,
WEXMASTER, WIDED, WIDGET_*, XANIMATE, XBACKREGISTER,
XBM_EDIT, XDISPLAYFILE, XFONT, XINTERANIMATE, XLOADCT,
XMANAGER, XMANAGERTOOL, XMENU, XMNG_TMPL, XMTOOL, XNOTHING,
XPALETTE, XPDMENU, XREGISTERED, XSQ_TEST, XSURFACE, XVAREDIT
```

filein
Disables file input operations by disallowing use of the following routines:

```
GET_KBRD, OPENR, READ, READF, READU, READ_*, TAPRD
```

fileout
Disables file output operations by disallowing use of the following routines:

```
OPENW, PRINTF, TAPWRT, WEOF, WRITEU, WRITE_*
```

fileio
Disables file input and output operations by disallowing use of the following routines:

```
ASSOC, CLOSE, EOF, FILEPATH, FLUSH, FSTAT, GET_LUN, IOCTL,
OPENU, POINT_LUN, REWIND, SKIPF
```

linking
Disables calls from IDL to external code by disallowing use of the following routines:

```
CALL_EXTERNAL, LINKIMAGE
```

os
Disables operating system access by disallowing use of the following routines:

```
CD, CALL_FUNCTION, CALL_METHOD, CALL_PROCEDURE, DEFINE_KEY,
DELETE_SYMBOL, DELLOG, EXECUTE, FILEPATH, FINDFILE, GETENV,
```

```
GET_SYMBOL, POPD, PRINTD, PUSHD, SETENV, SETLOG, SET_SYMBOL,
SPAWN, TRNLOG
```

df
Disables all Scientific Data Format routines (`CDF_*, HDF_*, NCDF_*`).

The ION HTTP Tunnel Broker

Network *firewalls* work by isolating a network from the Internet as a whole and allowing only pre-specified network operations to take place. In many cases, this means that traffic between an internal network and the Internet must go through a single computer, which allows connections of specified types and denies other connections. Firewalls allow computers and data on the "inside" to be relatively safe from intrusion by outsiders, while allowing the inside computers to make connections with computers on the Internet via a set of relatively limited protocols, such as HTTP and FTP.

The ION client/server model is based on a persistent two-way socket connection between the client and server. Firewalls, in most cases, do not allow arbitrary processes to open socket connections to remote servers. Because ION communication is not based on any standard protocol, it may not be able to penetrate a firewall that allows only the standard proxy servers (HTTP, FTP), and other well-known protocols through.

ION Java applications running behind a firewall have little chance of obtaining a connection through the firewall. However, ION applets running in a web browser can take advantage of the HTTP connections provided by the browser and use them to "tunnel" through the firewall and communicate with an external ION Server. The ION HTTP Tunnel Broker provides all the functionality necessary for ION Applets to successfully tunnel across most firewalls.

Note Because the Java vitual machines used by most web browsers use the SOCKS protocol rather than HTTP proxies for HTTP connections, the ION Tunnel Broker may not work through firewalls that do not support SOCKS.

The Tunnel Model

The ION Tunnel model includes an HTTP communications layer on the ION Client that maintains a connection to an ION HTTP Tunnel Broker. The Broker manages a set of *peers* that communicate with and control ION Servers. HTTP requests sent from the client to the server are dispatched to the appropriate peer and peer responses are sent back to the client through the Broker in the form of an HTTP response.

Normally, when an ION client makes a connection, the ION Daemon sets up a direct socket connection between the client and an ION Server process.

In the Tunnel model, ION clients connect to the ION Tunnel Broker rather than to the ION Daemon. The Tunnel Broker requests that the ION Daemon start an ION Server process, and attaches a peer to the server. The peer then acts as the client for communication with the ION Server. The peer buffers server responses, packs them into HTTP messages, and sends them through the Tunnel Broker back to the ION client.

Using the Tunnel Broker

Using the ION Tunnel Broker is very simple. On the server side, you must ensure that the ION Tunnel Broker is running; see "The ION Tunnel Broker Daemon" on page 59 for details. On the client side, you have the option of specifying one of three connection types via the CONNECTION_TYPE parameter in an ION applet:

- `"HTTP_CON"` — Make only HTTP connections, using the The ION HTTP Tunnel Broker.
- `"SOCK_CON"` — Make only socket connections, using only the ION Daemon.
- `"BEST_CON"` — Attempt to make a socket connection. If a socket connection is not possible, attempt to make an HTTP connection. This is the default setting.

Since "BEST_CON" is the default, you do not need to add the CONNECTION_TYPE parameter at all if you want your ION Server to accept either socket or HTTP connections. See "Parameters Specified via PARAM Tags" on page 14 for details on other parameters related to the ION Tunnel Broker.

The ION Tunnel Broker Daemon

The ION Tunnel Broker Daemon must be running for ION to be able to use HTTP connections. There are three ways to start the ION Tunnel Broker:

1. By specifying the `-http` flag to the `iond` command. With this method, both the ION Daemon and the ION Tunnel Broker are started at the same time. You can also specify the `-httpport` and `-httplog` flags to specify ION Tunnel Broker options. See "The ION Daemon" on page 53 for details.

2. Using the `ion_httpd` command at the command line:

   `$ RSI_DIR/ionrelease/bin/ion_httpd [switches]`

   Command-line switches for the `ion_httpd` command are listed below.

3. On Windows NT systems, by using the ION Properties dialog, described in "ION Windows NT Utilities" on page 62.

ion_httpd Command Line Switches

The ION Tunnel Broker Daemon accepts the following command-line options:

-ionhost=hostname

Set this switch equal to the name of the host on which the ION Daemon is running. Note that the ION Tunnel Broker may be running on a different host than the ION Daemon and ION Server. Note, however, that Java applet security requires that the Tunnel Broker be run on the same machine that the ION Java classes were loaded from.

-ionport=N

Set this switch equal to the port number that the ION Daemon is listening to. Note that the ION Tunnel Broker must be listening to a different port than the ION Daemon.

-port=N
Set this switch to the port number that the ION HTTP Tunnel Broker should watch for connection requests. If you do not specify a value for the `httpport` switch, the ION Tunnel Broker watches port 9085.

-logfile=filename [Windows NT only]
Set this switch to the name of the file in which you wish to save informational messages from the ION Tunnel Broker. If no logfile is specified, messages will be written to the standard output. (Under Unix, you can create a log file by redirecting the output from `ion_httpd` to a log file of your choosing using the normal system output redirection mechanism.)

-maxpeer=N
Set this switch to the maximum number of ION Tunnel Broker peers that can be active at once. If *N* is 0 (zero, the default setting), maximum number of peers will be equal to the number of IDL licenses you have available.

-timeout=N
Set this switch to the number of minutes the ION Tunnel Broker HTTP peer should stay alive for without hearing from the client. A timeout is necessary to close Tunnel Broker peer processes that may be left running if a browser crashes or experiences some other error that disconnects the browser without shutting down the peer process. If *N* is 0 (zero) the peer will never time out.

ION Command-Line Utilities

ION provides a set of utility programs that allow you to check the status of the ION Daemon or HTTP Tunnel Broker, or to shut down the daemon. On Unix platforms, run the utility programs from the shell prompt. On Windows NT platforms, you must open a Command Prompt window to execute the programs from the command line. On Windows platforms, you can also use a set of dialogs that accomplish the same operations as the command-line versions; the dialogs are discussed in "ION Windows NT Utilities" on page 62.

Checking the Status of the ION Daemon

Use the `ionstat` utility to determine the current status of the ION Daemon and Tunnel Broker. The status report includes the start time of the daemon and information about clients currently connected to the ION Server.

Under Windows NT, you must open a Command Prompt window to execute the command. You can also use the ION Status dialog, described in "ION Windows NT Utilities" on page 62.

```
$ ionstat [-host=hostname] [-port=port]
```

where:

-host=hostname
Set this flag to the name of the host on which the ION Daemon is running. Unless the `-rutil` flag was set when the ION Daemon was started, `ionstat` requests are only accepted from the host on which the daemon is running.

-port=port
Set this flag to the port number of the port that the ION Daemon is watching.

Shutting Down the ION Daemon

Use the `iondown` utility to shut down the ION Daemon and Tunnel Broker. Under Windows NT, you must open a Command Prompt window to execute the command.

Note Under Windows NT, you will generally use the ION service rather than starting and stopping the ION Daemon manually. However, if you use the `iond` command to start the ION Daemon on your machine, you can use the `iondown` command to stop it. There is no option in the ION Properties dialog to start and stop the ION Daemon directly.

```
$ iondown [-force] [-host=hostname] [-port=port]
```

where:

-force
Set this flag to force the ION Daemon to shut down without prompting. If `-force` is not specified, `iondown` will prompt you before shutting down the daemon.

-host=hostname
Set this flag to the name of the host on which the ION Daemon is running. Unless the `-rutil` flag was set when the ION Daemon was started, `iondown` requests are only accepted from the host on which the daemon is running.

-port=port
Set this flag to the port number of the port that the ION Daemon is watching.

Installing the ION Daemon Service under Windows NT

Use the `ion_srvinst.exe` program to install, control, and check the status of the ION Daemon Windows NT service. If you chose to install the ION Daemon service during the ION installation process, you do not need to install it again.

Note You can also use the ION Properties dialog, described in "ION Windows NT Utilities" on page 62, to install and remove the ION Service.

To use `ion_srvinst`, open a Command Prompt window and enter:

```
c:\rsi\ion11\ion_srvinst [-install] [-remove]
                          [-start = auto|manual] [-iondir=iondir]
```

(assuming your ION installation is in `c:\rsi\ion11`), where:

-install
Set this flag to install the ION Daemon service into the system.

-remove
Set this flag to remove the ION Daemon service from the system.

-start=[auto|manual]
Set this flag to specify the *start type* of the service. If set to `auto`, the ION Daemon service will be started by the Windows system at startup. If set to `manual`, the ION Daemon service must be started through the Service Control Panel. See "Starting and Stopping the ION Daemon Service Manually" on page 62 for details. The default is `manual`. Note that this option is only valid when installing the ION Daemon service—the flag is ignored if the `-install` switch is not also specified.

-iondir=[directory]
Use this flag to specify the ion installation directory. Setting this flag will override any Windows registry entries and environment variable settings

If no flags are specified, `ion_srvinst` will print out the current status of the service. This includes the following information:

- The "Start type" (manual or automatic).
- The full path specification for the executable the service is using (for example: `d:\rsi\ion11\ion_srv.exe`).
- The account name the service runs under, if available.

Starting and Stopping the ION Daemon Service Manually

To start the ION Daemon service manually, do the following:

1. Open the Services Control Panel.
2. Select "ION Daemon" from the list and click "Start" or "Stop".
3. To toggle the service between automatic and manual startup modes, click "Startup" and select the desired startup mode from the dialog.

Note You can also use the ION Properties dialog, described in "ION Windows NT Utilities" on page 62, to start and stop the ION Service.

Figure 6-1 shows the Windows NT Services control panel with the ION Daemon selected.

ION Windows NT Utilities

ION for Windows NT includes two dialog-based utilities that duplicate the functionality of the command-line utilities discussed in "ION Command-Line Utilities" on page 60. Shortcuts to both utilities — `wionprop.exe` and `wionstat.exe` — are installed in the ION program group by the ION installation program. This section discusses the use of the Windows ION utilities.

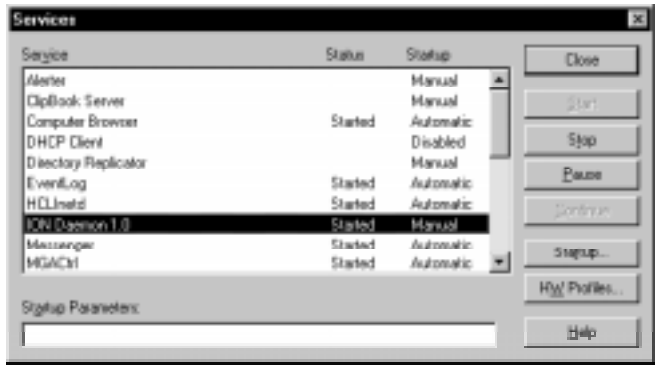


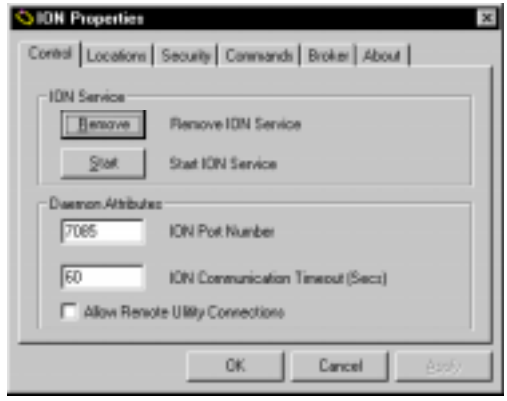
Figure 6-1: The Windows NT Services control panel.

The wionprop.exe Utility

Start the `wionprop.exe` utility by selecting "ION Properties" from the ION program group in the Windows Start menu. The utility presents a tabbed dialog that allows you to control ION settings. After changing a setting in the dialog, you can click "OK" to accept the change and close the dialog, "Apply" to accept the change but leave the dialog open, or "Cancel" to close the dialog without making any changes.

The ION Properties dialog has the following tabs:

Control Tab



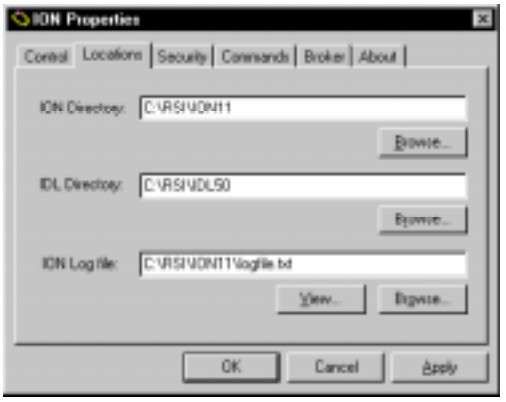
Click the "Remove"/"Install" button to remove or install the ION Service in the Windows service registry. If you installed the ION service during installation, you will probably not need to remove or install the service until a new version of ION is released. This button performs the same actions as the `ion_srvinst` utility routine.

Click the "Start"/"Stop" button to start or stop the ION service. This button performs the same actions as you would by manually starting and stopping the service using the Service Control Panel.

Set the value of the "ION Port Number" field to the port number the ION Service will listen on. Set the value of the "ION Communication Timeout" to the number of seconds ION will wait before closing a connection. Click the "Allow Remote Utility Connections" checkbox to allow the ION utility programs to control the ION Server from computers other than the one the server is running on. These three controls perform the same actions as the `-port`, `-timeout`, and `-rutil` flags to the iond command.

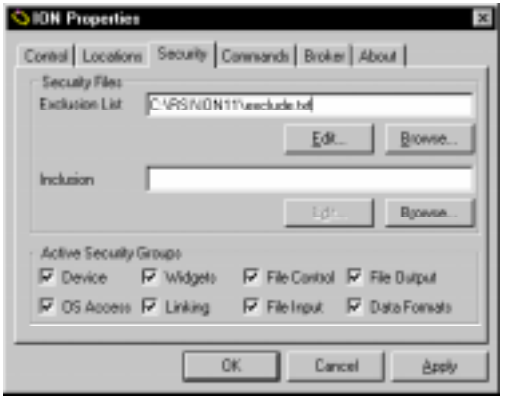
Locations Tab

Set the value of the "ION Directory" field to the location of the ION directory on your server machine. Set the value of the "IDL Directory" field to the location of IDL on your server machine. Set the value of the "ION Log File" field to the location of a text file that will contain the ION Server logs. These three fields perform the same actions as the `-iondir`, `-idldir`, and `-logfile` flags to the `iond` command. Click "View" to view the contents of the log file.



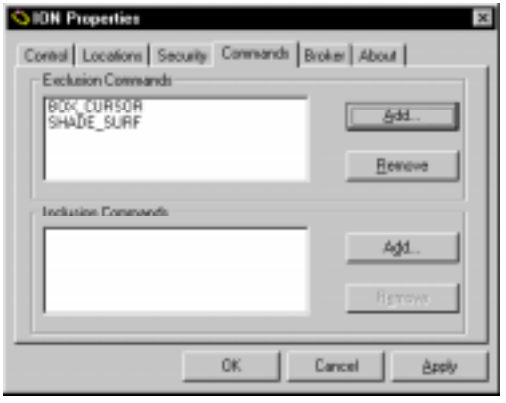
Security Tab

Set the value of the "Exclusion List" field to a text file that contains a list of commands ION should not execute. Set the value of the "Inclusion List" field to a text file that contains a list of commands ION is allowed to execute. Click the "Edit" button on either field to edit the text file. (See "Security Command Files" on page 53 for details.) Set the checkboxes in the "Active Security Groups" field to enable or disable entire classes of IDL functionality. The security groups are described in "Command Line Parameters" on page 54.

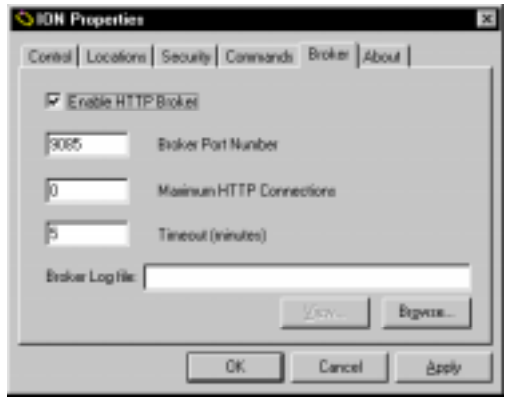


Commands Tab

Click "Add" to add the name of an IDL command to the list of individual commands to be excluded or allowed by the ION security mechanism. Select a command from either list and click "Remove" to remove that command from the list. (See "Security Command Files" on page 53 for details.)



Broker Tab



Click the "Enable HTTP Broker" checkbox to enable or disable the ION Tunnel Broker. Set the value of the "Broker Port Number" to the port number the ION Tunnel Broker will listen on. Set the value of the "Maximum HTTP Connections" field to the maximum number of HTTP connections allowed at any one time. Set the timeout value to the number of minutes an ION peer should wait before closing a connection. Set the value of the "Broker Log File" field to the location of a text file that will contain the ION Tunnel Broker logs. These three fields perform the same actions as the `-port`, `-maxpeer`, `-timeout`, and `-logfile` flags to the `ion_httpd` command. Click "View" to view the contents of the log file.

The wionstat.exe Utility

Start the `wionstat.exe` utility by selecting "ION Status" from the ION program group in the Windows Start menu. The ION Status dialog allows you to obtain information about the current state of an ION Daemon or ION Tunnel Broker. Set the value of the "Host" field to the name of the computer on which either the ION Daemon or ION Tunnel Broker is running. Set the "Port Number" field equal to the port being watched by either the ION Daemon or the ION Tunnel Broker. Click "Query" to retrieve information on the Daemon or Tunnel Broker running on the specified host and port. Click "Clear" to clear the display, or "OK" to dismiss the dialog.

The ION Server Process

Once the incoming client has been verified by the ION Daemon, the ION Daemon starts an ION Server process and connects the client with the ION Server process. The ION Server process checks out an ION license and then begins command processing. The ION Server process is responsible for the following:

- Reading requests from the ION client,

- Performing security checks on the client request,
- Executing valid ION/IDL commands,
- Sending graphic information and data to the ION client.

Security Checks

Once a command is received from the client, the request is passed through the ION security system. Any security failure causes the command to be logged and an error condition to be sent to the client. If the command passes the security system, it is passed to IDL for execution.

Command Execution

When a command is executed, all graphic and command log information is sent to the client. Once the command is completed the error status is sent to the client and the ION Server process waits for the next request.

Configuration Details

Very little configuration of your system is necessary to run ION — ION simply needs to know the locations of the ION directory and the associated IDL directory. Under Unix, environment variables specifying these locations are set automatically when the ION Daemon is started. Under Windows NT, the appropriate values are inserted in the Windows registry during the ION installation process.

You can override the default location settings using command-line switches to the ION Daemon (using either `iond` or `ion_srvinst`) described in "Command Line Parameters" on page 54, or you can set environment variables to override the defaults.

Note Under Unix, only the location of the ION directory can be overridden.

When ION runs, it uses the following algorithm to determine the locations of the ION and IDL directories:

1. If a command line switch was set, ION uses the value specified at the command line.
2. If the command-line switch is not set, ION will check the system environment for the directory environment variable. If the environment variable is present, its value will be used.
3. On a Windows NT system, if neither a command line switch or an environment variable is present, ION checks the Windows registry for the directory locations.

Environment Variables

Set the following environment variables to alter the default locations for ION and IDL.

ION_DIR

Set this environment variable to the directory that contains the ION distribution

IDL_DIR
Set this environment variable to the directory that contains the IDL distribution. Under Unix, the setting of this environment variable is ignored and its value set explicitly when the `iond` process starts.

Chapter 7

ION Class and Method Reference

The following topics are covered in this chapter:

This chapter describes the ION Java class library. Conventions used in this chapter are described below.

How to Use this Chapter

The elements of the ION Java class library are documented alphabetically in this chapter. The page or pages describing each class include a description of the class declaration, which provides pointers to the Java class (or other ION class) the class inherits from, if any. Note that this chapter does not provide documentation for the Java classes themselves; see your Java API reference materials for descriptions of the Java classes. Class methods are documented alphabetically (with the exception of the constructor method for the class, which is documented first) following the description of the class itself.

A description of each method follows its name. Beneath the general description of the method are a number of sections that describe the calling sequence for the method and its arguments (if any). These sections are described below.

Syntax

The "Syntax" section shows the proper syntax for calling the method.

Data Types

Java is a strongly-typed language, which means that input and output data variables must be created as or cast to the proper type before use. The "Syntax" description include the data type of each variable specified. For example, the following is a syntax description for the ION method that sets the value of an IDL variable:

```
setIDLVariable(String sName, IONVariable oVar)
```

In this case, there are two arguments to the `setIDLVariable` method: `sName` and `oVar`. The word "String" defines `sName` as a variable of type `string`. Similarly, the word "IONVariable" defines `oVar` as a variable of type `IONVariable`.

Multiple Syntax Definitions

Many ION Java methods can be called in more than one way. In these cases, all of the available syntax definitions are listed together. For example, the following are all valid ways to call the `setXValue` method of the IONContour class:

```
setXValue(int X[])

setXValue(float X[])

setXValue(double X[])

setXValue(String sName)
```

This means that the argument to the `setXValue` method can be either an integer, single-precision floating-point, or double-precision floating-point array, or a string value.

Optional Arguments

Arguments that are not required are included in the syntax definition enclosed in square brackets ([ ]). Do not confuse the use of square brackets to indicate that an argument is an array with the brackets that specify an optional argument. For example, the square brackets in this syntax definition indicate that the variable `X` is an array variable:

```
setXValue(int X[])
```

The square brackets in the following syntax definition indicate that the `portNumber` argument is optional:

```
connect(String hostname [, int portNumber])
```

Arguments

The "Arguments" section describes each valid argument to the method. Note that these arguments are positional parameters that must be supplied in the order indicated by the method's syntax.

Exceptions

The "Exceptions" section describes the ION exception values that are thrown when your error-handling code detects an error.

Example

The "Example" section includes, where appropriate, a short example showing the method in use.

Typographical Conventions

The following typographical conventions are used in this chapter:

- UPPER CASE
  IDL functions, procedures, and keywords are displayed in UPPER CASE type. For example, the calling sequence for an IDL procedure looks like this:

  CONTOUR, *Z [, X, Y]*

- Mixed Case
  ION object class and method names are displayed in Mixed Case type. Unlike IDL, the Java language is case-sensitive; names of ION Java methods and classes must be entered with the same capitalization as shown in this reference section.

- *Italic type*
  Arguments to ION procedures and functions — data or variables you must provide — are displayed in italic type.

- Square brackets ([ ])
  Square brackets used in calling sequences indicate that the enclosed arguments are optional. Do not type the brackets. In the above CONTOUR example, *X* and *Y* are optional arguments. Square brackets are also used to specify array elements.

- `Courier type`
  Names of ION classes and methods are displayed in `courier` type. Syntax descriptions are shown in **`courier bold`** and ***`bold italic`***. Examples are shown in `courier`.

Alphabetical List of Classes

The following table lists the methods for each ION Java class.

Class	Method
IONCallableClient	
	IONCallableClient()
	addIONCommandDoneListener()
	addIONDisconnectListener()
	addIONOutputListener()
	connect()
	disconnect()
	executeIDLCommand()
	getIDLVariable()
	removeIONCommandDoneListener()
	removeIONDisconnectListener()
	removeIONOutputListener()
	sendIDLCommand()
	setIDLVariable()
IONCanvas	
	IONCanvas()
	addIONMouseListener()
	getDownButtons()
	getMousePos()
	removeIONMouseListener()
IONCommandDoneListener	
	IONCommandComplete()
IONComplex	
	IONComplex()
	doubleValue()
	floatValue()

Class	Method
	getImaginary()
	getDImaginary()
	intValue()
	longValue()
	toString()
IONContour	
	IONContour()
	draw()
	getProperty()
	setProperty()
	setXValue()
	setYValue()
	setZValue()
IONDComplex	
	IONDComplex()
	doubleValue()
	floatValue()
	getImaginary()
	getDImaginary()
	intValue()
	longValue()
	toString()
IONDisconnectListener	
	IONDisconnection()
IONDrawable	
	createImage()
	flush()
	getGraphics()
	getImage()
	getIndex()
	getToolKit()
	initDrawable()

Class	Method
	isIndex()
	nColors()
	setIndex()
	size()
IONGraphicsClient	
	IONGraphicsClient()
	addIONDrawable()
	connect()
	copyArea()
	drawImage()
	drawLine()
	drawPolygon()
	drawText()
	erase()
	getCurrentIndex()
	getFreeIndex()
	getIONDrawableIndices()
	getNumIndices()
	readImage()
	removeIONDrawable()
	setDecomposed()
	setIONDrawable()
IONGrConnection	
	IONGrConnection()
	addDrawable()
	connect()
	debugMode()
	disconnect()
	executeIDLCommand()
	getIDLVariable()
	removeDrawable()
	setDrawable()

Class	Method
	setIDLVariable()
IONGrContour	
	IONGrContour()
	draw()
	getProperty()
	setProperty()
	setXValue()
	setYValue()
	setZValue()
IONGrDrawable	
	IONGrDrawable()
	addGraphic()
	draw()
	executeIDLCommand()
	getConnection()
	isConnected()
	removeGraphic()
	resetMulti()
	setMulti()
	setNoErase()
IONGrGraphic	
	IONGrGraphic()
	draw()
	getProperty()
	getPropertyString()
	registerProperty()
	setNoErase()
	setProperty()
IONGrPlot	
	IONGrPlot()
	draw()
	getProperty()

Class	Method
	setProperty()
	setXValue()
	setYValue()
IONGrSurface	
	IONGrSurface()
	draw()
	getProperty()
	setProperty()
	setXValue()
	setYValue()
	setZValue()
IONMouseListener	
	mouseMoved()
	mousePressed()
	mouseReleased()
IONOffScreen	
	IONOffScreen()
IONOutputListener	
	IONOutputText()
IONPaletteFilter	
	IONPaletteFilter()
	getColor()
	getIndexModel()
IONPlot	
	IONPlot()
	draw()
	getProperty()
	setProperty()
	setXValue()
	setYValue()
IONSurface	
	IONSurface()

Class	Method
	draw()
	getProperty()
	setProperty()
	setXValue()
	setYValue()
	setZValue()
IONVariable	
	IONVariable()
	arrayDimensions()
	getByte()
	getByteArray()
	getComplexArray()
	getDComplexArray()
	getDImaginary()
	getDouble()
	getDoubleArray()
	getFloat()
	getFloatArray()
	getImaginary()
	getInt()
	getIntArray()
	getShort()
	getShortArray()
	getString()
	getStringArray()
	isArray()
	toString()
	type()
IONWindow	
	IONWindow()
	setId()
	setOwner()

Class	Method
IONWindowingClient	
	IONWindowingClient()
	createPixmap()
	createWindow()
	deleteWindow()
	iconizeWindow()
	isPixmap()
	isWindow()
	showWindow()

IONCallableClient

The IONCallableClient class provides mechanisms to handle communication with the server, execution of IDL commands, retrieval of IDL command log output, and getting and setting IDL variables on the ION Server.

Class Declaration

```
public class IONCallableClient
```

Methods

- **IONCallableClient()**
  Construct an object of the IONCallableClient class.
- **addIONCommandDoneListener()**
  Add a "command done" listener to the client object.
- **addIONDisconnectListener()**
  Add a "disconnect" listener to the client object.
- **addIONOutputListener()**
  Add an "output" listener to the client object.
- **connect()**
  Connect to the server.
- **disconnect()**
  Shut down the ION Server and disconnect.
- **executeIDLCommand()**
  Execute an IDL command on the ION Server.
- **getIDLVariable()**
  Get the value of an IDL variable on the ION Server
- **removeIONCommandDoneListener()**
  Remove a "command done" listener from the client object.
- **removeIONDisconnectListener()**
  Remove a "disconnect" listener from the client object.
- **removeIONOutputListener()**
  Remove a listener from the client object.
- **setIDLVariable()**
  Set the value of an IDL variable on the ION Server.
- **sendIDLCommand()**
  Post an IDL command to the ION Server.

IONCallableClient()

The IONCallableClient() method constructs an IONCallableClient object. When it returns, all internal initialization is complete. No connection is made at this time.

Syntax

```
IONCallableClient()
```

Arguments

None.

Example

```
IONCallableClient client = new IONCallableClient();
```

addIONCommandDoneListener()

The addIONCommandDoneListener() method is used to register an object that implements the IONCommandDoneListener interface with this object. In order to provide support for mouse operations (the IDL cursor procedure) the main thread that handles the Java event loop must not block during IDL command execution. If the main event thread blocked and the IDL server requested a mouse location, the client and the server would be in a deadlock condition. To prevent a deadlock condition, this class provides the sendIDLCommand() method which sends the command to the server for execution and returns, not waiting for the command to complete. The class is informed of the commands completion status through the IONCommandDoneListener interface.

Syntax

```
addIONCommandDoneListener( IONCommandDoneListener listener)
```

Arguments

listener

An object that implements the IONCommandDoneListener interface. The *listener* is added to the internal listener list.

Exceptions

None.

addIONDisconnectListener()

The addIONDisconnectListener() method adds an object that implements the IONDisconnectListener interface to the internal list of registered listeners. When the client/server connection is disconnected, callback methods are called on the objects that are registered with the IONCallableClient.

Syntax

```
addIONDisconnectListener( IONDisconnectListener listener)
```

Arguments

listener

An object that implements the IONDisconnectListener interface. The *listener* is added to the internal listener list.

Exceptions

None.

addIONOutputListener()

The addIONOutputListener() method is used to add an object that implements the IONOutputListener interface to the internal list of listeners kept by this object. When any IDL output is sent from the server to the client, the output is sent to the objects contained in the listener list through the callback method defined by the IONOutputListener interface. This provides an efficient method of passing IDL output to the client and mimics the Java 1.1 event model.

Syntax

```
addIONOutputListener(IONOutputListener listener)
```

Arguments

listener

This is an object that implements the IONOuptutListener interface. This interface defines the format of the callback method used to pass IDL output to the listener object. The *listener* is added to the internal listener list.

Exceptions

None.

connect()

The connect() method establishes a connection between the client and the ION Server. The client and the server make validity checks and the communication protocol is established. If hostname and port information for both the ION Server and the ION HTTP Tunnel Broker are supplied, the connection type is set automatically to "BEST_CON". See setConnectionType() on page 85 for details on setting other connection types.

Syntax

```
connect(String hostname [, int portNumber])

connect(String hostname, int portNumber
       String httphost, int httpport)
```

Arguments

hostname

The name of the host that the ION Server or HTTP broker is running on. If the class is being created as part of a Java applet, most web browsers require that the host name be the same host that the applet is being served from. If the connection type is either "SOCK_CON" or "BEST_CON", this argument specifies the host that the ION Server is running on. If the connection type is "HTTP_CON" and the *hhtphost* argument is not specified, this agrument specifies the host that the HTTP Tunnel Broker is running on.

portNumber

The port number to use when connecting to the ION Server. If this number is not provided the default port number is used. If the connection type is either "SOCK_CON" or "BEST_CON", this argument specifies the port that the ION Server is running on. If the connection type is "HTTP_CON" and the *httpport* argument is not specified, this agrument specifies the port that the HTTP Tunnel Broker is running on.

httphost

The name of the host that the ION HTTP Tunnel Broker is running on. If all four arguments to the connect method are supplied, the connection type is automatically set to "BEST_CON".

httpport

The port number to use when connecting to the ION HTTP Tunnel Broker. If all four arguments to the connect method are supplied, the connection type is automatically set to "BEST_CON".

Exceptions

IOException

A network IO error detected

UnknownHostException
The given hostname is unknown

IONLicenseException
An ION license could not be obtained

disconnect()

Call the disconnect() method to shut down the ION Server (the daemon remains active), close the connection between the server and the client and free any resources that were being used by the connection. Once this method has been called, the object should be considered invalid and not used.

Syntax

```
disconnect()
```

Arguments

None.

Exceptions

None.

executeIDLCommand()

Use the executeIDLCommand() method to send an IDL command to the ION Server for execution. The function returns when the command is complete on the server.

Syntax

```
int iError = executeIDLCommand(String sCommand)
```

Return Value

The function returns the IDL system variable !ERROR

Arguments

sCommand

The IDL Command that is to be executed on the IDL server. The use of the "$" IDL command (to open a shell or command window) and the line continuation character ($) are prohibited (for security reasons, and because they can hang the server).

Exceptions

IOException

An error was detected during the IO operations used for communication.

IONIllegalCommandException

The specified IDL command was illegal (that is, it included the "$" character).

IONSecurityException

The specified IDL command is not allowed under the current ION security rules.

getConnectionType()

Use the getConnectionType() method to return the type of connection in use. See "The ION HTTP Tunnel Broker" on page 58 for details on connection types.

Syntax

```
int type = getConnectionType()
```

Return Value

The function returns on of the following values:

HTTP_CON — The client uses the ION HTTP Tunnel Broker exclusively.

SOCK_CON — The client uses a normal ION socket connection exclusively.

BEST_CON — The client makes the best connection it can.

These values are defined as constants in the IONCallableClient class defintion. The example below shows how to compare the returned value with the value defined in the IONCallableClient class.

Arguments

None.

Exceptions

None.

Example

To determine whether the connection in use is a socket-only connection, use a statement like the following:

```
if( getConnectionType() == IONCallableClient.SOCK_CON )
```

getIDLVariable()

Use the getIDLVariable() method to request the value of an IDL variable from the server. The value of the variable is then returned as and IONVariable object. If the variable does not exist on the server, it is created as an undefined type.

Syntax

```
IONVariable var = getIDLVariable(String sName)
```

Return Value

The function returns the value of the requested IDL variable in an IONVariable object.

Arguments

sName

The variable name whose value is desired.

Exceptions

IOException

An error was detected during the IO operations used for the client/server communication.

setConnectionType()

Use the setConnectionType() method to set the type of connection for the client. See "The ION HTTP Tunnel Broker" on page 58 for details on connection types.

Syntax

```
setConnectionType(int Type)
```

Arguments

Type

Set the Type argument to one of the three following values:

HTTP_CON — The client uses the ION HTTP Tunnel Broker exclusively.

SOCK_CON — The client uses a normal ION socket connection exclusively.

BEST_CON — The client makes the best connection it can.

These values are defined as constants in the IONCallableClient class defintion.

Exceptions

None.

Example

To set the connection type to HTTP-only, use the following statement:

```
setConnectionType(IONCallableClient.HTTP_ONLY)
```

removeIONCommandDoneListener()

Use the removeIONCommandDoneListener() method to remove an object that implements the IONCommandDoneListener interface from the list of listeners maintained by this object. If the listener is not contained in the internal list of listeners, the method returns silently.

Syntax

```
removeIONCommandDoneListener(IONCommandDoneListener
listener)
```

Arguments

listener

An object that implements the IONCommandDoneListener interface that is to be removed from the internal listener list.

Exceptions

None.

removeIONDisconnectListener()

Use the removeIONDisconnectListener() method to remove an object that implements the IONDisconnectListener interface from the internal Disconnect callback list. If the listener is not contained in the internal list of listeners, the method returns silently.

Syntax

```
removeIONDisconnectListener(IONDisconnectListener listener)
```

Arguments

listener

The object that implements an IONDisconnectListener interface that should be removed from the listener callback list.

Exceptions

None.

removeIONOutputListener()

Use the removeIONOutputListener() method to remove the given listener from the internal list of listeners. If the listener is not contained in the internal list of listeners, the method returns silently.

Syntax

```
removeIONOutputListener( IONOutputListener listener)
```

Arguments

listener

The object that implements an IONOutputListener interface that should be removed from the listener callback list.

Exceptions

None.

sendIDLCommand()

Use the sendIDLCommand() method to send an IDL command to the ION Server for execution. The IDL command is posted to the server for execution and the function returns. Notification of the commands completion is performed via the IONCommandDoneListener interface.

Syntax

```
sendIDLCommand(String sCommand)
```

Arguments

sCommand

The IDL Command that is to be executed on the IDL server. The use of the "$" IDL command (to open a shell or command window) and the line continuation character ($) are prohibited (for security reasons, and because they can hang the server).

Exceptions

IOException

An error was detected during the IO operations used for communication.

setIDLVariable()

Use the setIDLVariable() method to set the value of a variable in the ION Server. If the variable doesn't exist, it is created.

Syntax

```
setIDLVariable(String sName, IONVariable oVar)
```

Arguments

sName

The name of the variable to set on the server.

oVar

An object of type IONVariable that contains the value of the variable

Exceptions

IOException

An error was detected during the IO operations used for communication.

IONCanvas

Objects of the IONCanvas class represent a visible drawing area in which graphic output can be drawn.

Class Declaration

```
public class IONCanvas extends Canvas implements IONDrawable
```

Methods

- **IONCanvas()**
  Construct an object of the IONCanvas class.
- **addIONMouseListener()**
  Add a MouseListener object to the current canvas object.
- **getDownButtons()**
  Report position of the mouse cursor on the canvas object.
- **getMousePos()**
  Report mouse button status.
- **removeIONMouseListener()**
  Remove a MouseListener object from the current canvas object.

See also the descriptions of the IONDrawable interface class and the Java Canvas Class

IONCanvas()

The IONCanvas() method constructs an IONCanvas object of the given size. This object can then be placed in a Java AWT tree.

Syntax

```
IONCanvas( int width, int height)
```

Arguments

width

The width of the canvas.

height

The height of the canvas.

Exceptions

None.

addIONMouseListener()

Use the addIONMouseListener() method to set the object that implements the IONMouseListener interface as the current Mouse listener. When a mouse event of the requested type is detected, the mouse listener interface callback function is called. Only one mouse listener is active at one time. Note that only one mouse listener is allowed at a time. Any previously-set mouse listener is removed.

Syntax

```
addIONMouseListener(IONMouseListener listener, int request)
```

Arguments

listener

Object that implements the mouse listener interface request.

int

The mouse event type that is requested. This integer is a bit file that contains one or more of the ION_MOUSE_* codes that are define in the IONMouseListener interface.

Exceptions

None.

getMousePos()

The getMousePos() method returns the current location of the mouse cursor in the canvas.

Syntax

```
getMousePos()
```

Arguments

None.

Exceptions

None.

Example

```
Point pt = getMousePos();
```

getDownButtons()

The getDownButtons() method returns the current state of the mouse buttons in the canvas. The return value is a bit field where bit 1 is mouse button 1, bit 2 is mouse button 2 and bit three is mouse button 3. If a bit is set, the specific mouse button is down.

Syntax

```
getDownButtons()
```

Arguments

None.

Exceptions

None.

Example

```
int iState = getDownButtons();
```

removeIONMouseListener()

Use the removeIONMouseListener() method to remove a mouse listener from the object. If the given mouse listener is not the current listener, the function exits quietly.

Syntax

```
removeIONMouseListener(IONMouseListener listener)
```

Arguments

listener

The listener to remove.

Exceptions

None.

Example

```
removeIONMouseListener(listener);
```

IONCommandDoneListener

The IONCommandDoneListener interface class defines the method that an object must implement to receive notification that an IDL command has completed.

Class Declaration

```
public interface IONCommandDoneListener
```

Methods

- **IONCommandComplete()**
  Report on the status of a completed command.

IONCommandComplete()

Call the IONCommandComplete() method when a command that was sent to the IDL server is complete.

Syntax

```
public void IONCommandComplete(int iStatus, int iIDLStatus)
```

Arguments

iStatus

A value that indicates the status of the processing of the IDL command. This value is one of the following constants that are part of this class:

- ION_COMM_OK Command is OK
- ION_COMM_SECURITY Command security error.
- ION_COMM_INVALID Command was invalid

iIDLStatus

A value that indicates the success or failure of the execution of the IDL command. This is the value of !ERROR in the IDL session.

Exceptions

None.

Example

```
public void IONCommandComplete(iStatus, iIDLStatus);
```

IONComplex

The IONComplex class represents a complex number.

Class Declaration

```
public class IONComplex extends Number
```

Methods

- **IONComplex()**
  Construct an object of the IONComplex class.
- **doubleValue()**
  Returns the double value of the real portion of the number
- **floatValue()**
  Returns the float value of the number
- **getImaginary()**
  Returns the imaginary value of the number
- **getDImaginary()**
  Returns the imaginary value as a double.
- **intValue()**
  Returns the int value of the real portion of the number.
- **longValue()**
  Returns the long value of the real portion of the number.
- **toString()**
  Returns the string value of the real portion of the number.

IONComplex()

Use the IONComplex() method to construct an object of the IONComplex class.

Syntax

```
IONComplex(float r, float i)
```

Arguments

r
The real portion of the number.

i

The imaginary portion of the number.

Exceptions

None.

Example

```
IONComplex complexvar = new IONComplex(3.0, 2.0);
```

doubleValue()

The doubleValue() method returns the real portion of the complex number as a double-precision floating-point value.

Syntax

```
doubleValue()
```

Arguments

None.

Exceptions

None.

Example

```
double d = complexvar.doubleValue();
```

floatValue()

The floatValue() method returns the real portion of the complex number as a single-precision floating-point value.

Syntax

```
floatValue()
```

Arguments

None.

Exceptions

None.

Example

```
float f = complexvar.floatValue();
```

getImaginary()

The getImaginary() method returns the imaginary portion of the complex number as a single-precision floating-point value.

Syntax

```
getImaginary()
```

Arguments

None.

Exceptions

None.

Example

```
float i = complexvar.getImaginary();
```

getDImaginary()

The getDImaginary() method returns the imaginary portion of the complex number as a double-precision floating-point value.

Syntax

```
getDImaginary()
```

Arguments

None.

Exceptions

None.

Example

```
double d = complexvar.getDImaginary();
```

intValue()

The intValue() method returns the real portion of the complex number as an integer value.

Syntax

```
intValue()
```

Arguments

None.

Exceptions

None.

Example

```
int i = complexvar.intValue();
```

longValue()

The longValue() method returns the real portion of the complex number as a long-integer value.

Syntax

```
longValue()
```

Arguments

None.

Exceptions

None.

Example

```
long l = complexvar.longValue();
```

toString()

The toString() method returns the real portion of the complex number as a string value.

Syntax

```
toString()
```

Arguments

None.

Exceptions

None.

Example

```
String s = complexvar.toString();
```

IONContour

The IONContour class extends the IONGrDrawable class and contains an IONGrContour object to provide a easy way of drawing IDL contours. It can be inserted into an AWT tree.

Class Declaration

```
public class IONContour extends IONGrDrawable
```

Methods

- **IONContour()**
  Construct an object of the IONContour class.
- **draw()**
  Produces the output graphic and displays the graphic on the drawing surface of this class.
- **getProperty()**
  Used to get the value of a property.
- **setProperty()**
  Used to set a property for the graphic.
- **setXValue()**
  Sets the X value of the contour.
- **setYValue()**
  Sets the Y value of the contour.
- **setZValue()**
  Sets the Z data of the contour.

IONContour()

The IONContour() method constructs an object of the IONContour class.

Syntax

```
IONContour(int iWidth, int iHeight)
IONContour(int iWidth, int iHeight, int Z[][])
IONContour(int iWidth, int iHeight, float Z[][])
IONContour(int iWidth, int iHeight, double Z[][])
```

```
IONContour(int iWidth, int iHeight, int Z[])

IONContour(int iWidth, int iHeight, float Z[])

IONContour(int iWidth, int iHeight, double Z[])

IONContour(int iWidth, int iHeight, String sName)

IONContour(int iWidth, int iHeight, int Z[][], int X[],
    int Y[] )

IONContour(int iWidth, int iHeight, float Z[][], float X[],
    float Y[] )

IONContour(int iWidth, int iHeight, double Z[][],
    double X[], double Y[] )

IONContour(int iWidth, int iHeight, String sZName,
    String sXName, String sYName)
```

Arguments

iWidth
The width of the plot in pixels.

iHeight
The height of the plot in pixels.

Z
The Z values (data) to use in the contour.

sName, sZName
The name of the IDL variable to use for the Z (data) values of the contour.

X
An array holding the values for the X coordinates of the grid.

Y
An array holding the values for the Y coordinates of the grid.

sXName
The name of the IDL variable holding the values for the X coordinates of the grid.

sYName
The name of the IDL variable holding the values for the Y coordinates of the grid.

Exceptions

None.

draw()

Use the draw() method to produce and display a graphic in the drawing area that makes up this object.

Syntax

```
draw()
```

Arguments

None.

Exceptions

None.

getProperty()

Use the getProperty() method to get the current value of a property.

Syntax

```
getProperty(String Property)
```

Argument

Property

The name of the property

Properties Supported

The following IDL Contour properties are supported by IONContour.[get,set]Property. Refer to the IDL documentation on keywords available for use with the CONTOUR procedure for an explanation of each property:

C_ANNOTATION, C_CHARSIZE, C_COLORS, C_LABELS, C_LINESTYLE, C_ORIENTATION, C_SPACING, CLOSED, DOWNHILL, FILL, CELL_FILL, FOLLOW, IRREGULAR, LEVELS, NLEVELS, OVERPLOT, BACKGROUND, CHARSIZE, CLIP, COLOR, DATA, DEVICE, FONT, LINESTYLE, NOCLIP, NODATA, NOERASE, NORMAL, POSITION, SUBTITLE, T3D, TICKLEN, TITLE, MAX_VALUE, MIN_VALUE, NSUM, POLAR, XLOG, YNOZERO, YLOG, XCHARSIZE, YCHARSIZE, ZCHARSIZE, XGRIDSTYLE, YGRIDSTYLE, ZGRIDSTYLE, XMARGIN, YMARGIN, ZMARGIN, XMINOR, YMINOR, ZMINOR, XRANGE, YRANGE, ZRANGE, XSTYLE, YSTYLE, ZSTYLE, XTICKFORMAT, YTICKFORMAT, ZTICKFORMAT, XTICKLEN,

YTICKLEN, ZTICKLEN, XTICKNAME, YTICKNAME, ZTICKNAME, XTICKS, YTICKS, ZTICKS, XTICKV, YTICKV, ZTICKV, XTITLE, YTITLE, ZTITLE, ZVALUE, ZAXIS

Exceptions

None.

Example

```
IONVariable value = getProperty(Property);
```

setProperty()

Use the setProperty() method to set a property for the contour object.

Syntax

```
setProperty(String Property, IONVariable Value)
```

Arguments

Property

The name of the property to set.

Value

The value of the property.

Properties Supported

The following IDL Contour properties are supported by IONContour.[get,set]Property. Refer to the IDL documentation on keywords available for use with the CONTOUR procedure for an explanation of each property:

C_ANNOTATION, C_CHARSIZE, C_COLORS, C_LABELS, C_LINESTYLE, C_ORIENTATION, C_SPACING, CLOSED, DOWNHILL, FILL, CELL_FILL, FOLLOW, IRREGULAR, LEVELS, NLEVELS, OVERPLOT, BACKGROUND, CHARSIZE, CLIP, COLOR, DATA, DEVICE, FONT, LINESTYLE, NOCLIP, NODATA, NOERASE, NORMAL, POSITION, SUBTITLE, T3D, TICKLEN, TITLE, MAX_VALUE, MIN_VALUE, NSUM, POLAR, XLOG, YNOZERO, YLOG, XCHARSIZE, YCHARSIZE, ZCHARSIZE, XGRIDSTYLE, YGRIDSTYLE, ZGRIDSTYLE, XMARGIN, YMARGIN, ZMARGIN, XMINOR, YMINOR, ZMINOR, XRANGE, YRANGE, ZRANGE, XSTYLE, YSTYLE, ZSTYLE, XTICKFORMAT, YTICKFORMAT, ZTICKFORMAT, XTICKLEN, YTICKLEN, ZTICKLEN, XTICKNAME, YTICKNAME, ZTICKNAME, XTICKS,

YTICKS, ZTICKS, XTICKV, YTICKV, ZTICKV, XTITLE, YTITLE, ZTITLE, ZVALUE, ZAXIS

Exceptions

None.

setXValue()

Use the setXValue() method to reset the X value of the contour.

Syntax

```
setXValue(int X[])
setXValue(float X[])
setXValue(double X[])
setXValue(String sName)
```

Arguments

X

The new X value of the contour.

sName

The name of the IDL variable that contains the new X value of the contour

Exceptions

None.

setYValue()

Use the setYValue() method to reset the Y value of the contour.

Syntax

```
setYValue(int Y[])
setYValue(float Y[])
setYValue(double Y[])
```

setYValue(String ***sName*****)**

Arguments

Y

The new Y value of the contour

sName

The name of the IDL variable that contains the new Y value of the contour.

Exceptions

None.

setZValue()

Use the setZValue() method to reset the Z value of the contour.

Syntax

setZValue(int ***Z[]*****)**

setZValue(float ***Z[]*****)**

setZValue(double ***Z[]*****)**

setZValue(int ***Z[][]*****)**

setZValue(float ***Z[][]*****)**

setZValue(double ***Z[][]*****)**

setZValue(String ***sName*****)**

Argument

Z

The new Z value of the contour

sName

The name of the IDL variable that contains the new Z value of the contour.

Exceptions

None.

IONDComplex

The IONDComplex class represents a double-precision complex number.

Class Declaration

```
public classs IONComplex extends Number
```

Methods

- **IONDComplex()**
  Construct an object of the IONComplex class.
- **doubleValue()**
  Returns the double value of the real portion of the number
- **floatValue()**
  Returns the float value of the number
- **getImaginary()**
  Returns the imaginary value of the number
- **getDImaginary()**
  Returns the imaginary value as a double.
- **intValue()**
  Returns the int value of the real portion of the number.
- **longValue()**
  Returns the long value of the real portion of the number.
- **toString()**
  Returns the string value of the real portion of the number.

IONDComplex()

The IONDComplex() method constructs an object of the IONDComplex class.

Syntax

```
IONDComplex(double r, double i)
```

Arguments

r
The real portion of the number.

i
The imaginary portion of the number.

Exceptions

None.

Example

```
IONDComplex dcomplexvar = new IONDComplex(3.0, 2.0);
```

doubleValue()

The doubleValue() method returns the real portion of the complex number as a double-precision floating-point value.

Syntax

```
doubleValue()
```

Arguments

None.

Exceptions

None.

Example

```
double d = dcomplexvar.doubleValue();
```

floatValue()

The floatValue() method returns the real portion of the complex number as a single-precision floating-point value.

Syntax

```
floatValue()
```

Arguments

None.

Exceptions

None.

Example

```
float f = dcomplexvar.floatValue();
```

getImaginary()

The getImaginary() method returns the imaginary portion of the complex number as a single-precision floating-point value.

Syntax

```
getImaginary()
```

Arguments

None.

Exceptions

None.

Example

```
float i = dcomplexvar.getImaginary();
```

getDImaginary()

The getDImaginary() method returns the imaginary portion of the complex number as a double-precision floating-point value.

Syntax

```
getDImaginary()
```

Arguments

None.

Exceptions

None.

Example

```
double d = dcomplexvar.getDImaginary();
```

intValue()

The intValue() method returns the real portion of the complex number as an integer value.

Syntax

```
intValue()
```

Arguments

None.

Exceptions

None.

Example

```
int i = dcomplexvar.intValue();
```

longValue()

The longValue() method returns the real portion of the complex number as a long-integer value.

Syntax

```
longValue()
```

Arguments

None.

Exceptions

None.

Example

```
long l = dcomplexvar.longValue();
```

toString()

The toString() method returns the real portion of the complex number as a string value.

Syntax

```
toString()
```

Arguments

None.

Exceptions

None.

Example

```
String s = dcomplexvar.toString();
```

IONDisconnectListener

The IONDisconnectListener interface class defines a method that is called when the connection between the client and the server is disconnected. The reason for disconnection is defined by one of the constants that are a part of this interface.

Class Declaration

```
public interface IONDisconnectListener
```

Methods

- **IONDisconnection()**
  Report status when client and server are disconnected.

IONDisconnection()

The IONDisconnection() method is called when the connection between the client and the server is broken to report the reason for disconnection.

Syntax

```
IONDisconnection(int iStatus)
```

Arguments

iStatus

A constant (defined in the ION interface) corresponding to the reason for the disconnection. These constants are:

ION_DIS_OK	Normal disconnection due to disconnect() method being called.
ION_DIS_ERR	Disconnection caused by an error. Normally due to an interruption in the communication channel.
ION_DIS_SERVER	Disconnection due to server shutdown.

Exceptions

None.

IONDrawable

The IONDrawable interface class defines the methods that an object must implement to act as an ION drawable object. An IONDrawable is an object that can be drawn to by an IONGraphicsClient.

Class Declaration

```
public interface IONDrawable
```

Methods

- **createImage()**
  Used to create an offscreen image.
- **flush()**
  Forces all graphics operations to be displayed.
- **getGraphics()**
  Returns a Java graphics context for the device.
- **getImage()**
  Returns the image that is being drawn to.
- **getIndex()**
  Returns the IDL window index of the device
- **getIONGraphics()**
  Returns an ION graphics context for the device.
- **getToolKit()**
  Returns the Toolkit object for the system.
- **initDrawable()**
  A post-creation routine. This routine is called after the device has be created. This is needed because some graphics operations (creation of offscreen buffers) cannot be preformed in a constructor.
- **isIndex()**
  Returns true if the drawable is indexed color.
- **nColors()**
  Returns the number of colors in the device
- **setIndex()**
  Method used to set the IDL Window index in the device.
- **size()**
  Returns the size of the drawing area.

createImage()

Use the createImage() method to create an image from the given producer or of a given size.

Syntax

```
createImage(ImageProducer imProducer)
createImage(int width, int height)
```

Arguments

imProducer
Image producer that will create the image.

width
The width of the requested image.

height
The height of the requested image.

Exceptions

None.

Example

```
Image im = draw.createImage(imProducer);
Image im = draw.createImage(300, 300);
```

flush()

Use the flush() method to force all graphics operations to be displayed.

Syntax

```
flush()
```

Arguments

None.

Exceptions

None.

Example

```
draw.flush();
```

getGraphics()

Use the getGraphics() method to return a Graphics object that can be used to get graphics info on the Java drawing area or draw directly to it. Graphics in the Java drawing area are *not* accessible by IDL — use the getIONGraphics() method to get information or draw directly to a drawing buffer that is accessible by IDL.

Syntax

```
getGraphics()
```

Arguments

None.

Exceptions

None.

Example

```
Graphics g = draw.getGraphics();
```

getImage()

The getImage() method returns the image that contains the current state of the drawing area.

Syntax

```
getImage()
```

Arguments

None.

Exceptions

None.

Example

```
Image im = draw.getImage();
```

getIndex()

Use the getIndex() method to get the IDL Window index of the drawable. If no index has been set in the drawable, the getIndex() returns window index -1.

Syntax

`getIndex()`

Arguments

None.

Exceptions

None.

Example

```
int iIndex = draw.getIndex();
```

getIONGraphics()

Use the getIONGraphics() method to return a Graphics object that can be used to get graphics info on ION's drawing buffer or draw directly to it. Unlike the getGraphics() method, getIONGraphics() allows you to affect the actual IDL drawable area. For example, you would use the getIONGraphics() method when manipulating the buffer using the COPY keyword to IDL's DEVICE procedure.

Syntax

`getIONGraphics()`

Arguments

None.

Exceptions

None.

Example

```
Graphics g = draw.getIONGraphics();
```

getToolKit()

The getToolKit() method returns the toolkit for the system.

Syntax

```
getToolKit()
```

Arguments

None.

Exceptions

None.

Example

```
Toolkit tk = draw.getToolKit();
```

initDrawable()

Use the initDrawable() method to perform device initialization (such as the creation of off-screen images) that cannot be performed in the constructor method. initDrawable() is called after IONDrawable() has been called.

Syntax

```
initDrawable()
```

Arguments

None.

Exceptions

None.

Example

```
draw.initDrawable();
```

isIndex()

Use the isIndex() method to determine if the drawing area is an indexed color destination or not.

Syntax

```
inIndex()
```

Arguments

None.

Exceptions

None.

Example

```
boolean b = draw.isIndex();
```

nColors()

The nColors() method returns the number of colors available on the device.

Syntax

```
nColors()
```

Arguments

None.

Exceptions

None.

Example

```
int colors = draw.nColors();
```

setIndex()

The setIndex() method is used by the ION system to assign IDL drawable indices. Calling this method directly may cause the lists of drawable indices maintained by the server and client to become unsynchronized.

Syntax

```
setIndex(int iIndex)
```

Arguments

iIndex

The IDL window index for the object.

Exceptions

None.

Example

```
draw.setIndex(1);
```

size()

The size() method returns the size of the drawing area.

Syntax

```
size()
```

Arguments

None.

Exceptions

None.

Example

```
Dimension dim = draw.size();
```

IONGraphicsClient

The IONGraphicsClient class provides mechanisms to handle the processing of a graphic primitive data set from the IDL server. Information sent by the server is read by mechanisms provided by the super class IONCallableClient and dispatched to the handleServerAction() method of this class (this class overrides the handleServerAction() method).

Class Declaration

```
public class IONGraphicsClient extends IONCallableClient
    implements IONMouseListener
```

Methods

- **IONGraphicsClient()**
  Construct and object of the IONGraphicsClient class.
- **addIONDrawable()**
  Used to add an object that implements the IONDrawable interface (windows or off-screen images).
- **connect()**
  connects the client with the ION Server.
- **copyArea()**
  Copy an area from one drawable to another.
- **drawImage()**
  Draws an image on the current drawable
- **drawLine()**
  Draws a line on the current drawable
- **drawPolygon()**
  Draws a filled polygon on the current drawable
- **drawText()**
  Draws text on the current drawable.
- **erase()**
  erases the current drawable
- **getCurrentIndex()**
  Get the index of the current drawable.
- **getFreeIndex()**
  Get a free drawable index.

- **getIONDrawableIndices()**
  Get a list of assigned drawable indices.
- **getNumIndices()**
  Get the number of drawable indices allocated.
- **readImage()**
  Reads the current contents of the drawable
- **removeIONDrawable()**
  Used to remove an object from the internal list of IONDrawables maintained by this object.
- **setDecomposed()**
  Set decomposed mode on the connection.
- **setIONDrawable()**
  Set the current drawable.

IONGraphicsClient()

The IONGraphicsClient() method constructs an object of the IONGraphicsClient class.

Syntax

```
IONGraphicsClient()
```

Arguments

None.

Exceptions

None.

Example

```
IONGraphicsClient iclient = new IONGraphicsClient();
```

addIONDrawable()

Use the addIONDrawable() method to add an object that implements the IONDrawable interface to the internal list of drawing areas maintained by this object. An IONDrawable represents an area that graphic primitives can be rendered onto. When the window is

added to this class, that drawing area is made the current drawing area being used for graphical output. The developer has the option of telling ION what index to use and also requesting that the method send information about the new drawable to the server. The function returns the window index number that is used by IDL to reference the drawing area.

Syntax

```
addIONDrawable(IONDrawable drawable)
addIONDrawable(IONDrawable drawable, int index)
```

Arguments

drawable

An object that implements the IONDrawable interface.

index

The index to assign to the drawable. If no index is supplied, a free index is used.

Return Value

The function returns the window index number that is used by IDL to referance the drawable.

Exceptions

None.

Example

```
int iIndex = addIONDrawable( drawable);
int iIndex = addIONDrawable( drawable, index);
```

connect()

Use the connect() method to establish a connection between the client and the IDL server. The client and the server make validity checks and the communication protocol is established.

Syntax

```
connect(String hostname [, int portnumber])
```

Arguments

hostname
The name of the host that the ION Server is running on. If the class is being created as part of a Java applet, most web browsers require that the host name be the same host that the applet is being served from.

portNumber
The port number to use when connecting to the IDL server. If this number is not provided the default port number is used.

Exceptions

IOException
A network IO error detected

UnknownHostException
The given hostname is unknown

IONLicenseException
An IDL license could not be obtained

copyArea()

Use the copyArea() method to copy an area from one drawable to another.

Syntax

```
copyArea(int iSource, int iDest, int iXSrc, int iYSrc,
    int iWidth, int iHeight, int iXDest, int iYDest)
```

Arguments

iSource
The index of the source drawable.

iDest
The index of the destination drawable.

iXSrc, iYSrc
The lower left corner of the area to copy

iWidth, iHeight
The dimensions of the copy area

iXDest, iYDest
The location of the lower left corner of the copy area in the destination.

Exceptions
None.

drawImage()

Use the drawImage() method to draw an image on the current drawable.

Syntax

```
drawImage( byte bImage[], int iWidth, int iHeight,
    int iXoffset, int iYoffset)

drawImage( int iImage[], int iWidth, int iHeight,
    int iXoffset, int iYoffset)

drawImage(Image im, int iWidth, int iHeight,
    int iXoffset, int iYoffset)
```

Arguments

bImage
A byte array that contains an indexed color image.

iImage
An int array that contains an RGB image

im
A Java AWT Image

iWidth
The width of the image to draw

iHeight
The height of the image to draw

iXoffset
The X offset

iYoffset
The Y offset

Exceptions

None.

drawLine()

Use the drawLine() method to draw a line on the current drawable.

Syntax

```
drawLine( int iX[], int iY[], int nVerts, int iColor, int
iLinestyle)
```

Arguments

iX[]

Array of the X locations that make up the line

iY[]

Array of the Y locations that make up the line

nVerts

The number of vertices to draw.

iColor

The color to draw the line in.

iLinestyle

The IDL linestyle to use.

Exceptions

None.

drawPolygon()

Use the drawPolygon() method to draw a filled polygon on the current drawable.

Syntax

```
drawPolygon(int iX[], int iY[], int nVerts, int iColor)
```

Arguments

iX[]
The X points of the polygon

iY[]
The Y points of the polygon

nVerts
The number of vertices that make up the polygon.

iColor
The color to fill the polygon with.

Exceptions

None.

drawText()

Use the drawText() method to draw the given text using the provided font on the current drawable.

Syntax

```
drawText(String sText, int iX, int iY, int iColor,
    Font font)
```

Arguments

sText
The text to draw

iX
The x location of the text

iY
The y location of the text

iColor
The color to draw the text with.

font
The font to use for the text.

Exceptions

None.

erase()

Use the erase() method to erase the current drawable to the specified color.

Syntax

```
erase(int iColor)
```

Arguments

iColor

The color to erase the drawable to.

Exceptions

None.

getCurrentIndex()

Use the getCurrentIndex() method to get the index of the current drawable.

Syntax

```
getCurrentIndex()
```

Return Value

The current drawable index. If no drawable is current, -1 is returned.

Arguments

None.

Exceptions

None.

Example

```
int index = getCurrentIndex();
```

getFreeIndex()

The getFreeIndex() method returns the next available free drawable index greater than or equal to 32.

Syntax

```
int index = getFreeIndex()
```

Return Value

The next available index.

Arguments

None.

Exceptions

None.

Example

```
int index = getFreeIndex();
```

getIONDrawableIndices()

Use the getIONDrawableIndices() method to fill an array with the indices of the available drawables.

Syntax

```
getIONDrawableIndices(int[] iIndices)
```

Arguments

iIndices

An array of length getNumIndices that will be filled with the index values.

Exceptions

None.

getNumIndices()

The getNumIndices() method returns the number of drawable indices currently allocated.

Syntax

```
getNumIndices()
```

Return Value

The number of indices.

Arguments

None.

Exceptions

None.

Example

```
int num = getNumIndices();
```

readImage()

Use the readImage() method to read the contents of the current drawable.

Syntax

```
readImage()
readImage(int x0, int y0, int width, int height)
```

Arguments

x0

x start position of the rectangle to read.

y0
y start position of the rectangle to read.

width
The width of the rectangle to read.

height
The height of the rectangle to read.

Exceptions

None.

Example

```
Image im = readImage();
Image im = readImage( x0, y0, width, height);
```

removeIONDrawable()

Use the removeIONDrawable() method to remove an object that implements the IONDrawable interface from the internal list of IONDrawable objects.

Syntax

removeIONDrawable(IONDrawable ***drawable*****)**

removeIONDrawable(int ***index*****)**

Arguments

drawable
The drawable to remove.

index
The index of the drawable to remove.

Exceptions

None.

setDecomposed()

Use the setDecomposed() method to set an individual connection to the ION Server to use *decomposed color mode*. The decomposed color mode setting determines how ION will display graphics on a True color (24-bit or 32-bit color) device. If the argument is true (the default) a pixel value is treated as an RGB triplet. If the argument is false, the red component of the pixel is treated as an index into the current color table. For more information on decomposed color mode, see the documentation for the DECOMPOSED keyword to the DEVICE procedure in the IDL *Reference Guide*.

Once set, decomposed color mode applies to all drawables associated with a given connection.

Syntax

```
setDecomposed(boolean bDecomposed)
```

Arguments

bDecomposed

If bDecomposed is set to True, pixel values are interpreted as RGB triplets. (This is the default behavior.) If bDecomposed is set to False, the first eight bits of the pixel value (the red portion) are used as an index value into the currently loaded IDL color table.

Exceptions

None.

setIONDrawable()

Use the setIONDrawable() method to select which IONDrawable to use from the internal list of IONDrawable objects.

Syntax

```
setIONDrawable(int iIndex)
```

Arguments

iIndex

The index that was returned from the addIONDrawable() method when the object was added.

Exceptions

None.

IONGrConnection

The IONGrConnection class represents a connection between the client and the ION Server. It allows for the addition of multiple IONGrGraphic classes and has the primary function of acting as a communication module between the IONGrGraphic classes and the ION Server.

Class Declaration

```
public class IONGrConnection extends IONGraphicsClient
```

Methods

- **IONGrConnection()**
  Construct an object of the IONGrConnection class.
- **addDrawable()**
  Adds an IONGrDrawable class to this connection.
- **connect()**
  Connect with an ION Server.
- **debugMode()**
  Enable/Disable debug mode.
- **disconnect()**
  Disconnect with an ION Server.
- **executeIDLCommand()**
  Execute a given IDL command on the ION Server.
- **getIDLVariable()**
  Get the value of an IDL variable on the ION Server.
- **removeDrawable()**
  Removes an IONGrDrawable class from this connection.
- **setDrawable()**
  Sets the current drawable.
- **setIDLVariable()**
  Set the value of an IDL variable on the ION Server.

IONGrConnection()

The IONGrConnection() method constructs an object of the IONGrConnection class.

Syntax

```
IONGrConnection()
```

Arguments

None.

Example

```
IONGrConnection con = new IONGrConnection();
```

addDrawable()

Use the addDrawable() method to add the specified ION graphic to the connection object. Once added, the graphic can communicate with the ION Server and thus request graphics and information.

Syntax

```
addDrawable(IONGrDrawable ionGraphic)
```

Arguments

ionGraphic

An object of the IONGrDrawable class to add to the connection object.

Exceptions

None.

Example

```
IONGrDrawable draw;
con.addDrawable(draw);
```

connect()

Use the connect() method to connect to the ION Server.

Syntax

connect(String ***hostname*** **[,**int ***portnumber***]**)**

Arguments

hostname
A string containing the name of the host on which the ION Server is running.

portnumer
The portnumber of the host on which ION is running. If this number is not provided, the default ION port number is used.

Exceptions

IOException
A network I/O error was detected during connection.

UnknownHostException
The specified hostname is unknown.

IONLicenseException
The ION Server could not be licensed.

Example

```
try{
    con.connect(myhost);
}catch(IOException eIO) {
    System.err.println(“IO error”);
}catch(UnknownHostException eUH) {
    System.err.println(“Unknown Host error”);
}catch(IONLicenseException eIL) {
    System.err.println(“IDL License error”);
}
```

debugMode()

Use the debugMode() method to enable and disable the debug mode of the class. When debug mode is enabled, the command log output from the ION Server is displayed in a window when a Shift-click (shifted mouse button-press) event is detected on the drawing surface.

When debug mode is enabled, the class will buffer the output information for all registered drawables sent by the ION Server to the client class.

Syntax

```
debugMode(boolean enable)
```

Arguments

enable

If true, the debug mode is enabled, otherwise the debug mode is disabled.

Exceptions

None.

Example

```
con.debugMode(true);
```

disconnect()

Use the disconnect() method to close the connection between the client application and the ION Server. After a disconnect, no commands can be sent to the ION Server, but the component is repainted as necessary.

Syntax

```
disconnect()
```

Arguments

None.

Exceptions

None.

Example

```
con.disconnect();
```

executeIDLCommand()

Use the executeIDLCommand() method to send an IDL command to the ION Server for execution. Any graphical output resulting from the IDL command is displayed in the IONGrDrawable drawing area.

Syntax

```
int iError = executeIDLCommand(String sCommand)
```

Return Value

The function returns the IDL system variable !ERROR

Arguments

sIDLCommand

A string containing a valid IDL command.

Exceptions

IOException

Network communication error detected. Server is disconnected

IONIllegalCommandException

The specified IDL command was illegal.

IONSecurityException

The specified IDL command is not allowed under the current ION security rules.

Example

```
try{
    con.executeIDLCommand("PLOT, FINDGEN(10)");
}catch(IOException eIO) {
    System.err.println("IO error");
}catch(IONIllegalCommandException eIC) {
    System.err.println("Illegal Command error");
}catch(IONSecurityException eSE) {
    System.err.println("Security error");
}
```

getIDLVariable()

Use the getIDLVariable() method to get the value of an IDL variable from the ION Server.

Syntax

```
getIDLVariable(String name)
```

Arguments

name

A string containing the name of the variable to get from the ION Server.

Exceptions

IOException

A network error was detected while retrieving the variable.

Example

```
try{
    IDLVariable var = con.getIDLVariable(myvar);
}catch(IOException eIO) {
    System.err.println("IO error");
}
```

removeDrawable()

Use the removeDrawable() method to remove a graphic from the connection object. Once the graphic has been removed from the connection object, the graphic can no longer communicate with the ION Server.

Syntax

```
IONGrDrawable draw =
    removeDrawable(IONGrDrawable ionGraphic)

IONGrDrawable draw = removeDrawable(int iGraphic)
```

Return Value

A reference to the removed IONGrDrawable object.

Arguments

ionGraphic
An object of the IONGrDrawable class that is being removed from the connection.

iGraphic
A zero-based integer index designating which IONGrDrawable object to remove from the connection (the IDL window index).

Exceptions

None.

Example

```
IONGrDrawable draw = con.removeDrawable(iongraphic);
IONGrDrawable draw = con.removeDrawable(1);
```

setDrawable()

Use the setDrawable() method to designate which IONgrDrawable object will receive graphical output from the ION Server.

Syntax

```
boolean bsuccess = setDrawable(IONGrDrawable ionGraphic)
boolean bsuccess = setDrawable(int iGraphic)
```

Return Value

This routine returns False if the specified drawable is not registered with the connection, or True otherwise.

Arguments

ionGraphic
An instance of an IONGrDrawable object to set as the current drawable. This graphic must have been registered with the IONGrConnection object via the addDrawable() method.

iGraphic
A zero-based integer index designating which IONGrDrawable object to set as the current drawable. This graphic must have been registered with the IONGrConnection object via the addDrawable() method.

Exceptions

None.

Example

```
boolean bSuccess = con.setDrawable(ionGraphic);
boolean bSuccess = con.setDrawable(1);
```

setIDLVariable()

Use the setIDLVariable() method to set the value of a variable on the ION Server. If the variable is not present on the ION Server, a new variable is created.

Syntax

setIDLVariable(String ***sName*****,** IDLVariable ***oVar*****)**

Arguments

sName

A string containing the name of the variable to set.

oVar

The value the variable is to be set to.

Exceptions

IOException

An network error was detected.

Example

```
try{
    con.setIDLVariable(myvar, 10);
}catch(IOException eIO) {
    System.err.println("IO error");
}
```

IONGrContour

The IONGrContour class produces an IDL-generated contour in a drawing area. The class allows the user to enter data and set contour attributes at the program level.

Class Declaration

```
class IONGrContour extends IONGrGraphic
```

Methods

- **IONGrContour()**
  Construct an object of the IONGrContour class.
- **draw()**
  Produces the output graphic and displays the graphic on the drawing surface of this class.
- **getProperty()**
  Used to get the value of a property.
- **setProperty()**
  Used to set a property for the graphic.
- **setXValue()**
  Sets the X value of the contour.
- **setYValue()**
  Sets the Y value of the contour.
- **setZValue()**
  Sets the Z data of the contour

IONGrContour()

The IONGrContour() method constructs an IONGrContour object.

Syntax

```
IONGrContour()
IONGrContour(int Z[][])
IONGrContour(float Z[][])
IONGrContour(double Z[][])
IONGrContour(int Z[])
```

```
IONGrContour(float Z[])
IONGrContour(double Z[])
IONGrContour(String sName)
IONGrContour(int Z[], int X[], int Y[] )
IONGrContour(int Z[][], int X[], int Y[] )
IONGrContour(float Z[], float X[], float Y[] )
IONGrContour(float Z[][], float X[], float Y[] )
IONGrContour(double Z[], double X[], double Y[] )
IONGrContour(double Z[][], double X[], double Y[] )
IONGrContour(String sZName, String sXName, String sYName)
```

Arguments

Z
Z values (data) to use in the contour

sName, sZName
name of the IDL variable to use for the Z (data) values of the surface

X
array holding the values for the X coordinates of the grid.

Y
array holding the values for the Y coordinates of the grid.

sXName
name of the IDL variable holding the values for the X coordinates of the grid.

sYName
name of the IDL variable holding the values for the Y coordinates of the grid.

Exceptions

None.

draw()

Call the draw() method to produce and display a graphic in the drawing area that makes up this object.

Syntax

```
draw(IONGrConnection con)
```

Arguments

con

IONGrConnection used to issue the drawing commands to the server.

Exceptions

None.

getProperty()

Use the getProperty() method to get the current value of a property.

Syntax

```
getProperty(String Property)
```

Arguments

Property

The name of the property

Properties Supported

The following IDL Contour properties are supported by IONGrContour.[get,set]Property. Refer to the IDL documentation on keywords available for use with the CONTOUR procedure for an explanation of each property:

C_ANNOTATION, C_CHARSIZE, C_COLORS, C_LABELS, C_LINESTYLE, C_ORIENTATION, C_SPACING, CLOSED, DOWNHILL, FILL, CELL_FILL, FOLLOW, IRREGULAR, LEVELS, NLEVELS, OVERPLOT, BACKGROUND, CHARSIZE, CLIP, COLOR, DATA, DEVICE, FONT, LINESTYLE, NOCLIP, NODATA, NOERASE, NORMAL, POSITION, SUBTITLE, T3D, TICKLEN, TITLE, MAX_VALUE, MIN_VALUE, NSUM, POLAR, XLOG, YNOZERO, YLOG, XCHARSIZE, YCHARSIZE, ZCHARSIZE, XGRIDSTYLE, YGRIDSTYLE, ZGRIDSTYLE, XMARGIN, YMARGIN, ZMARGIN, XMINOR, YMINOR, ZMINOR, XRANGE, YRANGE, ZRANGE, XSTYLE, YSTYLE, ZSTYLE, XTICKFORMAT, YTICKFORMAT, ZTICKFORMAT, XTICKLEN, YTICKLEN, ZTICKLEN, XTICKNAME, YTICKNAME, ZTICKNAME, XTICKS, YTICKS, ZTICKS, XTICKV, YTICKV, ZTICKV, XTITLE, YTITLE, ZTITLE, ZVALUE, ZAXIS

Exceptions

None.

Example

```
IONVariable value = getProperty(Property);
```

setProperty()

Use the setProperty( ) method to set a property for the contour object.

Syntax

```
setProperty(String Property, IONVariable Value)
```

Arguments

Property

The name of the property to set.

Value

The value of the property

Properties Supported

The following IDL Contour properties are supported by IONGrContour.[get,set]Property. Refer to the IDL documentation on keywords available for use with the CONTOUR procedure for an explanation of each property:

C_ANNOTATION, C_CHARSIZE, C_COLORS, C_LABELS, C_LINESTYLE, C_ORIENTATION, C_SPACING, CLOSED, DOWNHILL, FILL, CELL_FILL, FOLLOW, IRREGULAR, LEVELS, NLEVELS, OVERPLOT, BACKGROUND, CHARSIZE, CLIP, COLOR, DATA, DEVICE, FONT, LINESTYLE, NOCLIP, NODATA, NOERASE, NORMAL, POSITION, SUBTITLE, T3D, TICKLEN, TITLE, MAX_VALUE, MIN_VALUE, NSUM, POLAR, XLOG, YNOZERO, YLOG, XCHARSIZE, YCHARSIZE, ZCHARSIZE, XGRIDSTYLE, YGRIDSTYLE, ZGRIDSTYLE, XMARGIN, YMARGIN, ZMARGIN, XMINOR, YMINOR, ZMINOR, XRANGE, YRANGE, ZRANGE, XSTYLE, YSTYLE, ZSTYLE, XTICKFORMAT, YTICKFORMAT, ZTICKFORMAT, XTICKLEN, YTICKLEN, ZTICKLEN, XTICKNAME, YTICKNAME, ZTICKNAME, XTICKS, YTICKS, ZTICKS, XTICKV, YTICKV, ZTICKV, XTITLE, YTITLE, ZTITLE, ZVALUE, ZAXIS

Exceptions

None.

setXValue()

Use the setXValue() method to reset the X value of the contour.

Syntax

```
setXValue(int X[])
setXValue(float X[])
setXValue(double X[])
setXValue(String sName)
```

Arguments

X

The new X value of the contour.

sName

The name of the IDL variable that contains the new X value of the surface

Exceptions

None

setYValue()

Use the setYValue() method to reset the Y value of the contour.

Syntax

```
setYValue(int Y[])
setYValue(float Y[])
setYValue(double Y[])
setYValue(String sName)
```

Arguments

Y

The new Y value of the contour

sName
The name of the IDL variable that contains the new Y value of the contour

Exceptions
None

setZValue()

Use the setZValue() method to reset the Z value of the contour.

Syntax
```
setZValue(int Z[])
setZValue(float Z[])
setZValue(double Z[])
setZValue(int Z[][])
setZValue(float Z[][])
setZValue(double Z[][])
setZValue(String sName)
```

Arguments
Z
The new Z value of the contour

sName
The name of the IDL variable that contains the new Z value of the contour

Exceptions
None

IONGrDrawable

Objects of the IONGrDrawable class represent a drawing area for IDL-produced graphics that can be part of a Java AWT. The IONGrDrawable can act alone as a drawing area or it can contain many IONGrGraphic objects. The way in which multiple Graphic objects are displayed in the drawable can be controlled using setNoErase() and setMulti().

Class Declaration

```
class IONGrDrawable extends IONCanvas
```

Methods

- **IONGrDrawable()**
  Construct an object of the IONGrDrawable class.
- **addGraphic()**
  Add a graphic object to be drawn.
- **draw()**
  Draw all graphic objects in the drawable.
- **executeIDLCommand()**
  Execute an IDL command on the ION Server.
- **getConnection()**
  Get the connection object associated with this server.
- **isConnected()**
  This method returns true if the drawable is associated with a server.
- **removeGraphic()**
  Remove a graphic object from the drawable.
- **resetMulti()**
  Reset "multi mode" to one visible drawable at a time.
- **sendIDLCommand()**
  Send an IDL command to the ION Server.
- **setMulti()**
  Specify how multiple graphic objects will be drawn in the server.
- **setNoErase()**
  Specify whether the drawable should be erased when new graphic is drawn.

IONGrDrawable()

The IONGrDrawable() method constructs an IONGrDrawable object of a specified size.

Syntax

```
IONGrDrawable(int iWidth, int iHeight)
```

Arguments

iWidth

The width of the drawing area.

iHeight

The height of the drawing area.

Exceptions

None.

addGraphic()

Use the addGraphic() method to add an IONGrGraphic to the drawable. Calling the draw() method causes all the graphics added in this manner to be displayed in the drawing area.

Syntax

```
addGraphic(IONGrGraphic ionGraphic)
```

Arguments

ionGraphic

graphic object to add.

Exceptions

None.

Example

```
addGraphic(ionGraphic);
```

draw()

Use the draw() method to draw all the graphics objects associated with this drawable. If there are no graphics objects associated, nothing happens.

Syntax

```
draw()
```

Arguments

None.

Exceptions

None.

executeIDLCommand()

Use the executeIDLCommand() method to send an IDL command to the ION Server for execution. Any resultant graphics is displayed in the IONGraphic drawing area.

Syntax

```
int iError = executeIDLCommand(String sCommand)
```

Return Value

The function returns the IDL system variable !ERROR

Arguments

sIDLCommand

A string containing a valid IDL command.

Exceptions

IOException

Network communication error detected. Server is disconnected

IONIllegalCommandException

The specified IDL command was illegal.

IONSecurityException

The specified IDL command is not allowed under the current ION security rules.

getConnection()

The getConnection() method returns the IONGrConnection object that this object is associated with. If no connection is associated with this object null is returned.

Syntax

```
getConnection()
```

Arguments

None.

Exceptions

None.

Example

```
IONGrConnection conn = getConnection();
```

isConnected()

The isConnected() method returns true if the Drawable is associated with a IONGrConnection.

Syntax

```
isConnected()
```

Arguments

None

Exceptions

None

Example

```
boolean connected = isConnected();
```

removeGraphic()

Use the removeGraphic() method to remove an IONGrGraphic from the drawable. The removeGraphic() method returns true on success or false if the specified graphic is not currently part of the system.

Syntax

```
removeGraphic(IONGrGraphic ionGraphic)
```

Arguments

ionGraphic

A graphic to remove from the drawable.

Exceptions

None.

Example

```
removeGraphic( ionGraphic);
```

resetMulti()

Use the resetMulti() method to reset the !P.multi system variable to 0 (one plot at a time, using the entire drawing area).

Syntax

```
resetMulti()
```

Arguments

None.

Exceptions

None.

sendIDLCommand()

Use the sendIDLCommand() method to send an IDL command to the ION Server for execution. The IDL command is posted to the server for execution and the function returns. Notification of the commands completion is performed via the IONCommandDoneListener interface.

Syntax

```
sendIDLCommand(String sCommand)
```

Arguments

sCommand

The IDL Command that is to be executed on the ION Server. The of the spawn command and the line continuation character ($) is prohibited (for security reasons, and because they can hang the server).

Exceptions

IOException

An error was detected during the IO operations used for communication.

setMulti()

Use the setMulti() method to set the !P.multi system variable that determines how multiple IDL plots or IONGrGraphics objects are displayed on the drawing area.

Syntax

```
setMulti(int iMulti[])
```

Arguments

iMulti

array defining the layout. See the IDL documentation for more information.

Exceptions

None.

setNoErase()

Use the setNoErase() method to specify whether or not the drawable should be erased between IONGrGraphic objects when the draw() method is called.

Syntax

```
setNoErase()
```

Arguments

bNoErase

if the drawing area should not be erased.

Exceptions

None.

IONGrGraphic

The IONGrGraphic abstract class implements methods that are used by sub-classes to manage and store properties.

Class Declaration

```
abstract class IONGrGraphic
```

Methods

- **IONGrGraphic()**
  Construct an object of the IONGrGraphic class.
- **draw()**
  the object.
- **getProperty()**
  the value of the given property.
- **getPropertyString()**
  a string that represents the properties. This string can then used with an IDL command.
- **setNoErase()**
  the object whether or not it should be erased when another object drawn.
- **setProperty()**
  the value of a property in the property list.
- **registerProperty()**
  a valid property.

IONGrGraphic()

The IONGrGraphic() method constructs an object of the IONGrGraphic class.

Syntax

```
IONGrGraphic()
```

Arguments

None.

Exceptions

None.

draw()

The draw() method is defined by sub-classes to issue the appropriate IDL command to draw the graphic object.

Syntax

```
draw(IONGrConnection con)
```

Arguments

con

IONGrConnection used to issue the drawing commands to the server.

Exceptions

None.

Example

```
draw(con);
```

getProperty()

The getProperty() method returns the value of a property. The property is returned as an object. It is the responsibility of the caller to cast the object to the correct type.

Syntax

```
getProperty(String sProperty)
```

Arguments

sProperty

The name of the property.

Exceptions

None.

Example

```
IONVariable = getProperty(sProperty);
```

getPropertyString()

The getPropertyString() method returns a string that contains the values of all the properties contained in the object. The string is formatted such that each property name makes up an IDL keyword and the value of the property is the value of the keyword. This string can be appended to an IDL graphics command string.

Note This is a protected method, and can only be accessed from objects that subclass the IONGrGraphic class.

Syntax

```
getPropertyString()
```

Arguments

None.

Exceptions

None.

Example

```
String sProperties = getPropertyString();
```

setNoErase()

The setNoErase() method is defined by subclasses to set the appropriate property for the graphic object that corresponds to the concept of 'no erase'.

Syntax

```
setNoErase(boolean bFlag)
```

Arguments

bFlag
if the object is not to be erased when other objects are drawn.

Exceptions

None.

Example

```
setNoErase(bFlag);
```

setProperty()

The setProperty() method is used to set the value of a property in the objects property list. If the property already exists in the property list, it's value is replaced, otherwise the property is added to the property list.

Syntax

setProperty(String ***sProperty*****,** IONVarible ***value*****)**

Arguments

sProperty
The name of the property to set.

value
The value of the property. This must be an object or an array.

Exceptions

None.

Example

```
void setProperty(sProperty, value);
```

registerProperty()

The registerProperty() method is used to register a property name as being valid. When the setProperty() or getProperty() methods are called, they check the validity of the object against the list of valid properties.

Note This is a protected method, and can only be accessed from objects that subclass the IONGrGraphic class.

Syntax

```
registerProperty(String PropertyName)
```

Arguments

PropertyName

Name of the property.

Exceptions

None.

Example

```
protected registerProperty(PropertyName);
```

IONGrPlot

The IONGrPlot class produces an IDL generated plot in a drawing area. The class allows the user to enter data and plot attributes at the program level.

Class Declaration

```
class IONGrPlot extends IONGrGraphic
```

Methods

- **IONGrPlot()**
  Construct an object of the IONPlot class.
- **draw()**
  Produces the output graphic and displays the graphic on the drawing surface of this class.
- **getProperty()**
  Used to get the value of a property.
- **setProperty()**
  Used to set a property for the graphic.
- **setXValue()**
  Sets the X value of the plot.
- **setYValue()**
  Sets the Y value of the plot.

IONGrPlot()

The IONGrPlot() method constructs an object of the IONGrPlot class.

Syntax

```
IONGrPlot()
IONGrPlot(int X[] [, int Y[]])
IONGrPlot(float X[] [, float Y[]])
IONGrPLot(double X[] [, double Y[]])
IONGrPLot(String sXName)
IONGrPlot(String sXName, String sYname)
```

Arguments

X
X values of the plot

Y
Y values of the plot

sXName
The name of an IDL variable to use for the X values in this plot.

sYName
The name of an IDL variable to use for the Y values in this plot.

Exceptions

None.

draw()

Use the draw() method to display the plot in the drawing area that makes up this object.

Syntax

```
draw(IONGrConnection con)
```

Arguments

con
IONGrConnection used to issue the drawing commands to the server.

Exceptions

None.

Example

```
draw(con);
```

getProperty()

Use the getProperty() method to get the current value of a property.

Syntax

```
getProperty(String Property)
```

Arguments

Property

The name of the property

Properties Supported

The following IDL Plot properties are supported by IONGrPlot.[get,set]Property. Refer to the IDL documentation on keywords available for use with the PLOT procedure for an explanation of each property:

BACKGROUND, CHARSIZE, CLIP, COLOR, DATA, DEVICE, FONT, LINESTYLE, NOCLIP, NODATA, NOERASE, NORMAL, POSITION, PSYM, SUBTITLE, SYMSIZE, T3D, TICKLEN, TITLE, MAX_VALUE, MIN_VALUE, NSUM, POLAR, XLOG, YNOZERO, YLOG, ZLOG

Exceptions

IONInvalidPropertyException

if the property is not valid for the object.

Examples

```
IONVariable value = getProperty(Property);
```

setProperty()

Use the setProperty() method to set a property for the plot object.

Syntax

```
setProperty(String Property, IONVariable Value)
```

Arguments

Property

The name of the property to set.

Value

The value of the property

Properties Supported

The following IDL Plot properties are supported by IONGrPlot.[get,set]Property. Refer to the IDL documentation on keywords available for use with the PLOT procedure for an explanation of each property:

BACKGROUND, CHARSIZE, CLIP, COLOR, DATA, DEVICE, FONT, LINESTYLE, NOCLIP, NODATA, NOERASE, NORMAL, POSITION, PSYM, SUBTITLE, SYMSIZE, T3D, TICKLEN, TITLE, MAX_VALUE, MIN_VALUE, NSUM, POLAR, XLOG, YNOZERO, YLOG, ZLOG

Exceptions

None.

Example

```
setProperty(Property, Value);
```

setXValue()

Use the setXValue() method to reset the X value of the plot

Syntax

```
setXValue(int X[])
setXValue(float X[])
setXValue(double X[])
setXValue(String sName)
```

Arguments

X

The new X value of the plot

sName

The name of an IDL variable to use for the X value.

Exceptions

None.

setYValue()

Use the setYValue() method to reset the Y value of the plot

Syntax

```
setYValue(int Y[])
setYValue(float Y[])
setYValue(double Y[])
setYValue(String sName)
```

Arguments

Y

The new Y value of the plot

sName

The name of the IDL variable to use for the Y value.

Exceptions

None.

IONGrSurface

The IONGrSurface class produces an IDL-generated surface in a drawing area. The class allows the user to enter data and set surface attributes at the program level.

Class Declaration

```
class IONGrSurface extends IONGrGraphic
```

Methods

- **IONGrSurface()**
  Construct an object of the IONGrSurface class.
- **draw()**
  Produces the output graphic and displays the graphic on the drawing surface of this class.
- **getProperty()**
  Used to get the value of a property.
- **setProperty()**
  Used to set a property for the graphic.
- **setXValue()**
  Sets the X value of the surface.
- **setYValue()**
  Sets the Y value of the surface.
- **setZValue()**
  Sets the Z data of the surface

IONGrSurface()

The IONGrSurface() method constructs an object of the IONGrSurface class.

Syntax

```
IONGrSurface()
IONGrSurface(int Z[][])
IONGrSurface(float Z[][])
IONGrSurface(double Z[][])
IONGrSurface(String sName)
```

```
IONGrSurface(int Z[][], int X[], int Y[] )
IONGrSurface(float Z[][], float X[], float Y[] )
IONGrSurface(double Z[][], double X[], double Y[] )
IONGrSurface(String sZName, String sXName, String sYName)
```

Arguments

Z
Z (data) values for the surface

sName, sZName
name of the IDL variable to use for the Z (data) of the surface

X
array holding the values for the X coordinates of grid.

Y
array holding the values for the Y coordinates of grid.

sXName
name of the IDL variable holding the values for X coordinates of the grid.

sYName
name of the IDL variable holding the values for Y coordinates of the grid.

Exceptions

None.

draw()

Use the draw() method to display the surface in the drawing area that makes up this object.

Syntax

```
draw(IONGrConnection con)
```

Arguments

con
IONGrConnection used to issue the drawing commands to the server.

Exceptions

None.

getProperty()

Use the getProperty() method to get the current value of a property.

Syntax

```
getProperty(String Property)
```

Arguments

Property

The name of the property

Properties Supported

The following IDL Surface properties are supported by IONGrSurface.[get,set]Property. Refer to the IDL documentation on keywords available for use with the SURFACE procedure for an explanation of each property:

AX, AZ, BOTTOM, HORIZONTAL, LEGO, LOWER_ONLY, SAVE, SHADES, UPPER_ONLY, ZAXIS, BACKGROUND, CHARSIZE, CLIP, COLOR, DATA, DEVICE, FONT, LINESTYLE, NOCLIP, NODATA, NOERASE, NORMAL, POSITION, SUBTITLE, T3D, TICKLEN, TITLE, MAX_VALUE, MIN_VALUE, NSUM, POLAR, XLOG, YNOZERO, YLOG, XCHARSIZE, YCHARSIZE, ZCHARSIZE, XGRIDSTYLE, YGRIDSTYLE, ZGRIDSTYLE, XMARGIN, YMARGIN, ZMARGIN, XMINOR, YMINOR, ZMINOR, XRANGE, YRANGE, ZRANGE, XSTYLE, YSTYLE, ZSTYLE, XTICKFORMAT, YTICKFORMAT, ZTICKFORMAT, XTICKLEN, YTICKLEN, ZTICKLEN, XTICKNAME, YTICKNAME, ZTICKNAME, XTICKS, YTICKS, ZTICKS, XTICKV, YTICKV, ZTICKV, XTITLE, YTITLE, ZTITLE, ZVALUE, ZLOG

Exceptions

None.

Example

```
IONVariable value = getProperty(Property);
```

setProperty()

Use the setProperty() method to set a property for the surface object.

Syntax

```
setProperty(String Property, IONVarible Value)
```

Arguments

Property

The name of the property to set.

Value

The value of the property

Properties Supported

The following IDL Surface properties are supported by IONGrSurface.[get,set]Property. Refer to the IDL documentation on keywords available for use with the SURFACE procedure for an explanation of each property:

AX, AZ, BOTTOM, HORIZONTAL, LEGO, LOWER_ONLY, SAVE, SHADES, UPPER_ONLY, ZAXIS, BACKGROUND, CHARSIZE, CLIP, COLOR, DATA, DEVICE, FONT, LINESTYLE, NOCLIP, NODATA, NOERASE, NORMAL, POSITION, SUBTITLE, T3D, TICKLEN, TITLE, MAX_VALUE, MIN_VALUE, NSUM, POLAR, XLOG, YNOZERO, YLOG, XCHARSIZE, YCHARSIZE, ZCHARSIZE, XGRIDSTYLE, YGRIDSTYLE, ZGRIDSTYLE, XMARGIN, YMARGIN, ZMARGIN, XMINOR, YMINOR, ZMINOR, XRANGE, YRANGE, ZRANGE, XSTYLE, YSTYLE, ZSTYLE, XTICKFORMAT, YTICKFORMAT, ZTICKFORMAT, XTICKLEN, YTICKLEN, ZTICKLEN, XTICKNAME, YTICKNAME, ZTICKNAME, XTICKS, YTICKS, ZTICKS, XTICKV, YTICKV, ZTICKV, XTITLE, YTITLE, ZTITLE, ZVALUE, ZLOG

Exceptions

None.

setXValue()

Use the setXValue() method to reset the X value of the surface

Syntax

```
setXValue(int X[])
```

setXValue(float ***X[]*****)**

setXValue(double ***X[]*****)**

setXValue(String ***sName*****)**

Arguments

X

The new X value of the surface

sName

The name of the IDL variable that contains the new X value of the surface

Exceptions

None.

Examples

setYValue()

Use the setYValue() method to reset the Y value of the surface

Syntax

setYValue(int ***Y[]*****)**

setYValue(float ***Y[]*****)**

setYValue(double ***Y[]*****)**

setYValue(String ***sName*****)**

Arguments

Y

The new Y value of the surface.

sName

The name of the IDL variable that contains the new Y value of the surface.

Exceptions

None.

setZValue()

Use the setZValue() method to reset the Z value of the surface

Syntax

```
setZValue(int Z[][])
setZValue(float Z[][])
setZValue(double Z[][])
setZValue(String sName)
```

Arguments

Z

The new Z value of the surface

sName

The name of the IDL variable that contains the new Z value of the surface

Exceptions

None.

IONMouseListener

The IONMouseListener interface class defines the callback methods an object must define to be notified of mouse events occurring on an object that implements the IONDrawable interface.

Class Declaration

```
public interface IONMouseListener
```

Methods

- **mousePressed()**
  Called when a mouse button down event occurred.
- **mouseMoved()**
  Called when the mouse moved.
- **mouseReleased()**
  Called when a mouse button up event occurred.

mouseMoved()

Call the mouseMoved() method when a mouse cursor is moved in a drawable. Note that the Mouse Listener must have been registered in the drawable prior to calling mouseMoved().

Syntax

```
mouseMoved(IONDrawable drawable, int X, int Y, long when,
    int mask)
```

Arguments

drawable

The IONDrawable object that the event occurred in.

X

The X location of the mouse

Y

The Y location of the mouse

when
The time when the event happened.

mask
Current mouse button state.

Exceptions
None

mousePressed()

Call the mousePressed() method when a mouse button is pressed in a drawable. Note that the Mouse Listener must have been registered in the drawable prior to calling mousePressed().

Syntax
```
mousePressed(IONDrawable drawable, int X, int Y, long when,
    int mask)
```

Arguments
drawable
The IONDrawable object that the event occurred in.

X
The X location of the mouse.

Y
The Y location of the mouse.

when
the time when the event happened.

mask
Which button was pressed.

Exceptions
None

Example

mouseReleased()

Call the mouseReleased() method when a mouse button is released in a drawable. Note that the Mouse Listener must have been registered in the drawable prior to calling mouseReleased().

Syntax

```
mouseReleased(IONDrawable drawable, int X, int Y, long when,
    int mask)
```

Arguments

drawable
The IONDrawable object that the event occurred in.

X
The X location of the mouse

Y
The Y location of the mouse

when
The time when the event happened.

mask
Current mouse button state. The left mouse button is represented by 1 (one), the middle mouse button by 2, and the right mouse button by 4.

Note In Unix versions of Java, it is impossiible to determine which mouse button was released if more than one button was pressed before the button release. As a result, on Unix platforms ION reports the following button release events:

Buttons Pressed	Button Release Reported by ION
left and middle	left
left and right	left
middle and right	right
left, middle, and right	left

Button release events are reported correctly in Windows versions of Java.

Exceptions

None.

IONOffScreen

Objects of the IONOffScreen class represent an invisible drawing area that graphic output can be placed on.

Class Declaration

```
public class IONOffScreen implements IONDrawable
```

Methods

- **IONOffScreen()**
  Construct an object of the IONOffScreen class.

  See also the descriptions of the IONDrawable interface class.

IONOffScreen()

The IONOffScreen() method constructs an object of the IONOffscreen class.

Syntax

```
IONOffScreen( int width, int height, Component comp)
```

Arguments

width
The width of the drawing area

height
The height of the drawing area

comp
A visible used to create images. This needs to be a component that is already visible on the users screen in order for the OffScreen to be properly created.

Exceptions

None

Example

```
IONOffScreen offscreen = new IONOffScreen();
```

IONOutputListener

The IONOutputListener interface class defines the method that an object must implement to receive ION Server output text.

Class Declaration

```
public interface IONOutputListener
```

Methods

- **IONOutputText()**
  Retrieve a line of text from the ION Server.

IONOutputText()

The IONOutputText() method is called when a line of output text is available from the ION Server.

Syntax

```
IONOutputText(String sLine)
```

Arguments

sLine

A line of output text from the ION Server.

Exceptions

None.

IONPaletteFilter

The IONPaletteFilter class is used to implement an ION version of a Java ImageFilter. This image filter is used to change the color palette (table) being used. The Java Image scheme uses image filter object to change attributes and perform operations on images and as such the IONPaletteFilter is used to change the color table of indexed color systems.

Class Declaration

```
public class IONPaletteFilter extends ImageFilter
```

Methods

- **IONPaletteFilter()**
  Construct an object of the IONPaletteFilter class.
- **getColor()**
  Returns a Color object given an index into the color palette.
- **getIndexModel()**
  Returns the IndexColorModel object for this filter

IONPaletteFilter()

The IONPaletteFilter() method constructs an object of the IONPaletteFilter class with a color Palette that contains the passed in red, green and blue values

Syntax

```
IONPaletteFilter(byte r[], byte g[], byte b[])
```

Arguments

r

The red values for the color palette

g

The green values for the color palette

b

The blue values for the color palette.

Exceptions

None.

getColor()

The getColor() method returns a Color object that represents the color at the given index into the color palette of the filter.

Syntax

```
getColor(int index)
```

Arguments

index

The index into the color palette. The red, green and blue values at this location are used to construct the Color object.

Exceptions

None.

Example

```
Color color = getColor(index);
```

getIndexModel()

The getIndexModel() method returns the current IndexColorModel that is being used by the filter

Syntax

```
getIndexModel()
```

Arguments

None.

Exceptions

None.

Example

```
IndexColorModel icm = getIndexModel();
```

IONPlot

The IONPlot class extends the IONGrDrawable class and contains an IONGrPlot to provide a easy way of drawing IDL plots. It can be inserted into an AWT tree.

Class Declaration

```
public class IONPlot extends IONGrDrawable
```

Methods

- **IONPlot()**
  Construct an object of the IONPlot class.
- **draw()**
  Produces the output graphic and displays the graphic on the drawing surface of this class.
- **getProperty()**
  Used to get the value of a property.
- **setProperty()**
  Used to set a property for the graphic.
- **setXValue()**
  Sets the X value of the plot.
- **setYValue()**
  Sets the Y value of the plot.

IONPlot()

The IONPlot() method constructs an object of the IONPlot class.

Syntax

```
IONPlot(int iWidth, int iHeight)
IONPlot(int iWidth, int iHeight, int X[] [, int Y[]])
IONPlot(int iWidth, int iHeight, float X[] [, float Y[]])
IONPlot(int iWidth, int iHeight, double X[] [, double Y[]])
IONPlot(int iWidth, int iHeight, String sXName)
IONPlot(int iWidth, int iHeight, String sYName)
```

Arguments

iWidth
The width of the plot.

iHeight
The height of the plot.

X
The X values of the plot

Y
The Y values of the plot

sXName
The name of an IDL variable to use for the X values of this plot.

sYName
The name of an IDL variable to use for the Y values of this plot.

Exceptions

None.

draw()

Use the draw() method to produce and display a graphic in the drawing area that makes up this object.

Syntax

```
draw()
```

Arguments

None.

Exceptions

None.

getProperty()

Use the getProperty() method to get the current value of a property.

Syntax

```
getProperty(String Property)
```

Arguments

Property

The name of the property

Properties Supported

The following IDL Plot properties are supported by IONPlot.[get,set]Property. Refer to the IDL documentation on keywords available for use with the Plot procedure for an explanation of each property:

BACKGROUND, CHARSIZE, CLIP, COLOR, DATA, DEVICE, FONT, LINESTYLE, NOCLIP, NODATA, NOERASE, NORMAL, POSITION, PSYM, SUBTITLE, SYMSIZE, T3D, TICKLEN, TITLE, MAX_VALUE, MIN_VALUE, NSUM, POLAR, XLOG, YNOZERO, YLOG, ZLOG

Exceptions

None.

Example

```
IONVariable value = getProperty(Property);
```

setProperty()

Use the setProperty() method to set a property for the plot object.

Syntax

```
setProperty(String Property, IONVariable Value)
```

Arguments

Property

The name of the property to set.

Value
The value of the property

Properties Supported
The following IDL Plot properties are supported by IONPlot.[get,set]Property. Refer to the IDL documentation on keywords available for use with the Plot procedure for an explanation of each property:

BACKGROUND, CHARSIZE, CLIP, COLOR, DATA, DEVICE, FONT, LINESTYLE, NOCLIP, NODATA, NOERASE, NORMAL, POSITION, PSYM, SUBTITLE, SYMSIZE, T3D, TICKLEN, TITLE, MAX_VALUE, MIN_VALUE, NSUM, POLAR, XLOG, YNOZERO, YLOG, ZLOG

Exceptions
None.

setXValue()

Use the setXValue() method to reset the X value of the plot

Syntax
```
setXValue(int X[])
setXValue(float X[])
setXValue(double X[])
setXValue(String sName)
```

Arguments
X
The new X value of the plot

sName
The name of an IDL variable to use for the X value.

Exceptions
None

setYValue()

Use the setYValue() method to reset the Y value of the plot

Syntax

```
setYValue(int Y[])
setYValue(float Y[])
setYValue(double Y[])
setYValue(String sName)
```

Argument

Y

The new Y value of the plot

sName

The name of the IDL variable to use for the Y value.

Exceptions

None

IONSurface

The IONSurface class extends the IONGrDrawable class and contains an IONGrSurface object to provide a easy way of drawing IDL surfaces. It can be inserted into an AWT tree.

Class Declaration

```
public class IONSurface extends IONGrDrawable
```

Methods

- **IONSurface()**
  Construct an object of the IONSurface class.
- **draw()**
  Produces the output graphic and displays the graphic on the drawing surface of this class.
- **getProperty()**
  Used to get the value of a property.
- **setProperty()**
  Used to set a property for the graphic.
- **setXValue()**
  Sets the X value of the surface.
- **setYValue()**
  Sets the Y value of the surface.
- **setZValue()**
  Sets the Z data of the surface

IONSurface()

Use the IONSurface() method to construct an object of the IONSurface class.

Syntax

```
IONSurface(int iWidth, int iHeight)
IONSurface(int iWidth, int iHeight, int Z[][])
IONSurface(int iWidth, int iHeight, float Z[][])
IONSurface(int iWidth, int iHeight, double Z[][])
IONSurface(int iWidth, int iHeight, String sName)
```

```
IONSurface(int iWidth, int iHeight, int Z[][], int X[],
    int Y[] )

IONSurface(int iWidth, int iHeight, float Z[][], float X[],
    float Y[] )

IONSurface(int iWidth, int iHeight, double Z[][],
    double X[], double Y[] )

IONSurface(int iWidth, int iHeight, String sZName,
    String sXName, String sYName)
```

Arguments

iWidth
The width of the plot.

iHeight
The height of the plot.

Z
The Z (data) values for the surface

sName, sZName
The name of the IDL variable to use for the Z (data) values of the surface

X
An array holding the values for the X coordinates of the grid.

Y
An array holding the values for the Y coordinates of the grid.

sXName
The name of the IDL variable holding the values for the X coordinates of the grid.

sYName
The name of the IDL variable holding the values for the Y coordinates of the grid.

Exceptions

None.

draw()

Use the draw() method to produce and display a graphic in the drawing area that makes up this object.

Syntax

```
draw()
```

Arguments

None

Exceptions

None.

getProperty()

Use the getProperty() method to get the current value of a property.

Syntax

```
getPropety(String Property)
```

Arguments

Property

The name of the property

Properties Supported

The following IDL Surface properties are supported by IONSurface.[get,set]Property. Refer to the IDL documentation on keywords available for use with the SURFACE procedure for an explanation of each property:

AX, AZ, BOTTOM, HORIZONTAL, LEGO, LOWER_ONLY, SAVE, SHADES, UPPER_ONLY, ZAXIS, BACKGROUND, CHARSIZE, CLIP, COLOR, DATA, DEVICE, FONT, LINESTYLE, NOCLIP, NODATA, NOERASE, NORMAL, POSITION, SUBTITLE, T3D, TICKLEN, TITLE, MAX_VALUE, MIN_VALUE, NSUM, POLAR, XLOG, YNOZERO, YLOG, XCHARSIZE, YCHARSIZE, ZCHARSIZE, XGRIDSTYLE, YGRIDSTYLE, ZGRIDSTYLE, XMARGIN, YMARGIN, ZMARGIN, XMINOR, YMINOR, ZMINOR, XRANGE, YRANGE, ZRANGE, XSTYLE, YSTYLE, ZSTYLE, XTICKFORMAT, YTICKFORMAT, ZTICKFORMAT, XTICKLEN, YTICKLEN, ZTICKLEN, XTICKNAME, YTICKNAME, ZTICKNAME, XTICKS, YTICKS, ZTICKS, XTICKV, YTICKV, ZTICKV, XTITLE, YTITLE, ZTITLE, ZVALUE, ZLOG

Exceptions

None.

Example

```
IONVariable value = getProperty(Property);
```

setProperty()

Use the setProperty() method to set a property for the plot object.

Syntax

```
setProperty(String Property, IONVariable Value)
```

Arguments

Property

The name of the property to set.

Value

The value of the property

Properties Supported

The following IDL Surface properties are supported by IONSurface.[get,set]Property. Refer to the IDL documentation on keywords available for use with the SURFACE procedure for an explanation of each property:

AX, AZ, BOTTOM, HORIZONTAL, LEGO, LOWER_ONLY, SAVE, SHADES, UPPER_ONLY, ZAXIS, BACKGROUND, CHARSIZE, CLIP, COLOR, DATA, DEVICE, FONT, LINESTYLE, NOCLIP, NODATA, NOERASE, NORMAL, POSITION, SUBTITLE, T3D, TICKLEN, TITLE, MAX_VALUE, MIN_VALUE, NSUM, POLAR, XLOG, YNOZERO, YLOG, XCHARSIZE, YCHARSIZE, ZCHARSIZE, XGRIDSTYLE, YGRIDSTYLE, ZGRIDSTYLE, XMARGIN, YMARGIN, ZMARGIN, XMINOR, YMINOR, ZMINOR, XRANGE, YRANGE, ZRANGE, XSTYLE, YSTYLE, ZSTYLE, XTICKFORMAT, YTICKFORMAT, ZTICKFORMAT, XTICKLEN, YTICKLEN, ZTICKLEN, XTICKNAME, YTICKNAME, ZTICKNAME, XTICKS, YTICKS, ZTICKS, XTICKV, YTICKV, ZTICKV, XTITLE, YTITLE, ZTITLE, ZVALUE, ZLOG

Exceptions

None.

setXValue()

Use the setXValue() method to reset the X value of the surface

Syntax

```
setXValue(int X[])
setXValue(float X[])
setXValue(double X[])
setXValue(String sName)
```

Arguments

X

The new X value of the surface

sName

The name of the IDL variable that contains the new X value of the surface

Exceptions

None.

setYValue()

Use the setYValue() method to reset the Y value of the surface

Syntax

```
setYValue(int Y[])
setYValue(float Y[])
setYValue(double Y[])
setYValue(String sName)
```

Arguments

Y

The new Y value of the surface.

sName

The name of the IDL variable that contains the new Y value of the surface.

Exceptions

None.

setZValue()

Use the setZValue() method to reset the Z value of the surface

Syntax

```
setZValue(int Z[])
setZValue(float Z[])
setZValue(double Z[])
setZValue(String sName)
```

Arguments

Z

The new Z value of the surface

sName

The name of the IDL variable that contains the new Z value of the surface

Exceptions

None.

IONVariable

Objects of the IONVariable class provide a client-side representation of an IDL variable. IONVariable objects are used to read and write data between the IDL server and clients.

Class Declaration

```
public class IONVariable
```

Constants

The following constants are used to identify data types

Type	Description
TYPE_UNDEFINED	Variable is of IDL type undefined
TYPE_BYTE	Variable is of IDL type byte
TYPE_INT	Variable is of IDL type int
TYPE_LONG	Variable is of IDL type long
TYPE_FLOAT	Variable is of IDL type float
TYPE_DOUBLE	Variable is of IDL type double
TYPE_STRING	Variable is of IDL type string
TYPE_COMPLEX	Variable is of IDL type complex
TYPE_DCOMPLEX	Variable is of IDL type double complex

Methods

- **IONVariable()**
  Construct an object of the IONVariable class.
- **arrayDimensions()**
  Returns an int array that contains the arrays dimensions
- **getByte()**
  Returns the byte value of the variable
- **getByteArray()**
  Returns the byte array of the variable
- **getComplexArray()**
  Returns the array of IONComplex values.
- **getDComplexArray()**
  Returns the array of IONDComplex values.

- **getDImaginary()**
  Returns the imaginary value of a double complex variable
- **getDouble()**
  Returns the double value of the variable
- **getDoubleArray()**
  Returns the double array value of the variable
- **getFloat()**
  Returns the float value of the variable.
- **getFloatArray()**
  Returns the float array of the variable
- **getImaginary()**
  Returns the imaginary value of a complex variable
- **getInt()**
  Returns the int value of the variable
- **getIntArray()**
  Returns the int array of the variable
- **getShort()**
  Returns the short value of the variable.
- **getShortArray()**
  Returns the short array value of the variable
- **getString()**

  Returns the string value of the variable
- **getStringArray()**

  Returns the string array value of the variable
- **isArray()**
  Returns true of the variable is an array.
- **toString()**
  Returns a string that represents the variable value.
- **type()**

  Returns the type of the variable

IONVariable()

The IONVariable() method constructs an object of the specified IDL data type. The variable can be either a scalar or an array.

Syntax

Scalars

```
IONVariable()
IONVariable(byte b)
IONVariable(short s)
IONVariable(int i)
IONVariable(float f)
IONVariable(double d)
IONVariable(String s)
IONVariable(IONComplex cmp)
IONVariable(IONDComplex dcmp)
```

Arrays

```
IONVariable(byte b[], int dims[])
IONVariable(short s[], int dims[])
IONVariable(int i[], int dims[])
IONVariable(float f[], int dims[])
IONVariable(double d[], int dims[])
IONVariable(String s[], int dims[])
IONVariable(IONComplex cmp[], int dims[])
IONVariable(IONDComplex dcmp[], int dims[])
```

Arguments

Most arguments are straightforward. If no arguments are specified, the IONVariable object corresponds to an IDL variable of type “Undefined”. Type “short” corresponds to IDL type “integer”, and type “int” corresponds to IDL type “long integer”. The size of arrays and the array dimension array are determined through the use of the Java array length property.

Example

To create an IONVariable object of type float:

```
IONVariable oVariable = new IONVariable(1234.5678);
```

To create an IONVariable object of type float array of size (100,100,3):

```
float[] farr = new float[100*100*3];
int dims[] = new int[3];
dims[0] = 100;
dims[1] = 100;
dims[2] = 3;
oVariable = new IONVariable(farr, dims);
```

arrayDimensions()

The arrayDimensions() method returns an int array that contains the size of the dimensions of the array variable. If the variable is not an array, an exception is thrown.

Syntax

```
int dims[] = arrayDimensions()
```

Return Value

The function returns an int array that contains the size of each dimension in the corresponding element of the array. The number of dimensions available can be determined through the length property of the returned array.

Arguments

None.

Exceptions

IONNotAnArrayException

Thrown if the IONVariable is not an array.

Example

```
try {
```

```
int dims[] = arrayDimensions();
}catch(IONNotAnArrayException e){
   system.err.println("Variable is not an array");
}
```

getByte()

The getByte() method returns the byte value of the variable. If the value is not of type byte, the scalar value is converted to a byte.

Syntax

```
byte b = getByte()
```

Return Value

The method returns the byte value of the variable.

Arguments

None.

Exceptions

IONIsAnArrayException

Thrown if the variable contains an array value

NumberFormatException

Thrown if the variable is a string that cannot be converted.

Example

```
try {
byte b = getByte();
}catch(IONIsAnArrayException e){
   system.err.println("Variable is an array");
}catch(NumberFormatException e){
   system.err.println("String Cannot be converted");
}
```

getByteArray()

The getByteArray() method returns the byte array value of the variable. If the value is not of type byte, a conversion is attempted.

Syntax

```
byte b[] = getByteArray()
```

Return Value

The method returns the byte array value of the variable

Arguments

None.

Exceptions

IONNotAnArrayException

Thrown if the variable is not an array

Example

```
try {
byte b[] = getByteArray();
}catch(IONNotAnArrayException e){
   system.err.println("Variable is not an array");
}
```

getComplexArray()

The getComplexArray() method returns the value of the complex array variable.

Syntax

```
IONComplex c[] = getComplexArray()
```

Return Value

The method returns the value of the complex array variable

Arguments

None.

Exceptions

IONNotAnArrayException

Thrown if the variable is not an array

Example

```
try {
IONComplex c[] = getComplexArray();
}catch(IONNotAnArrayException e){
   system.err.println("Variable is not an array");
}
```

getDComplexArray()

The getDComplexArray() method returns the value of the double complex array variable. If the value is not of type double complex, conversion is supplied.

Syntax

```
IONDComplex dc[] = getDComplexArray()
```

Return Value

The method returns the value of the double complex array variable

Arguments

None.

Exceptions

IONNotAnArrayException

Thrown if the variable is not an array

Example

```
try {
IONDComplex dc[] = getDComplexArray();
```

```
}catch(IONNotAnArrayException e){
   system.err.println("Variable is not an array");
}
```

getDImaginary()

The getDImaginary() method returns the imaginary value of the double complex variable. If the value is not of type double complex, zero is returned.

Syntax

```
double i = getDImaginary()
```

Return Value

The method returns the imaginary value of the double complex variable.

Arguments

None.

Exceptions

IONIsAnArrayException

Thrown if the variable contains an array value

Example

```
try {
double i = getDImaginary();
}catch(IONIsAnArrayException e){
   system.err.println("Variable is an array");
}
```

getDouble()

The getDouble() method returns the double value of the variable. If the value is not of type double, the scalar value is converted to a double.

Syntax

```
double d = getDouble()
```

Return Value

The method returns the double value of the variable.

Arguments

None.

Exceptions

IONIsAnArrayException

Thrown if the variable contains an array value

NumberFormatException

Thrown if the variable is a string and cannot be converted.

Example

```
try {
double d = getDouble();
}catch(IONIsAnArrayException e){
   system.err.println("Variable is an array");
}catch(NumberFormatException e){
   system.err.println("String Cannot be converted");
}
```

getDoubleArray()

The getDoubleArray() method returns the double array value of the variable. If the value is not of type double, a conversion is attempted.

Syntax

```
double d[] = getDoubleArray()
```

Return Value

The method returns the double array value of the variable

Arguments

None.

Exceptions

IONNotAnArrayException

Thrown if the variable is not an array

Example

```
try {
double d[] = getDoubleArray();
}catch(IONNotAnArrayException e){
   system.err.println("Variable is not an array");
}
```

getFloat()

The getFloat() method returns the float value of the variable. If the value is not of type float, the scalar value is converted to a float.

Syntax

```
float f = getFloat()
```

Return Value

The method returns the float value of the variable.

Arguments

None.

Exceptions

IONIsAnArrayException

Thrown if the variable contains an array value

NumberFormatException

Thrown if the variable is a string and cannot be converted.

Example

```
try {
float f = getFloat();
}catch(IONIsAnArrayException e){
   system.err.println("Variable is an array");
}catch(NumberFormatException e){
   system.err.println("String Cannot be converted");
}
```

getFloatArray()

The getFloatArray() method returns the float array value of the variable. If the value is not of type float, a conversion is attempted.

Syntax

```
float f[] = getFloatArray()
```

Return Value

The method returns the float array value of the variable

Arguments

None.

Exceptions

IONNotAnArrayException

Thrown if the variable is not an array

Example

```
try {
float f[] = getFloatArray();
}catch(IONNotAnArrayException e){
   system.err.println("Variable is not an array");
}
```

getImaginary()

The getImaginary() method returns the imaginary value of the complex variable. If the value is not of type complex, zero is returned.

Syntax

```
float i = getImaginary()
```

Return Value

The method returns the imaginary value of the complex variable.

Arguments

None.

Exceptions

IONIsAnArrayException

Thrown if the variable contains an array value

Example

```
try {
float i = getImaginary();
}catch(IONIsAnArrayException e){
   system.err.println("Variable is an array");
}
```

getInt()

The getInt() method returns the int value of the variable. If the value is not of type int (IDL type long), the scalar value is converted to a int.

Syntax

```
int i = getInt()
```

Return Value

The method returns the int value of the variable.

Arguments

None.

Exceptions

IONIsAnArrayException

Thrown if the variable contains an array value

NumberFormatException

Thrown if the variable is a string and cannot be converted.

Example

```
try {
int i = getInt();
}catch(IONIsAnArrayException e){
   system.err.println("Variable is an array");
}catch(NumberFormatException e){
   system.err.println("String Cannot be converted");
}
```

getIntArray()

The getIntArray() method returns the int array value of the variable. If the value is not of type int, a conversion is attempted.

Syntax

```
int i[] = getIntArray()
```

Return Value

The method returns the int array value of the variable

Arguments

None.

Exceptions

IONNotAnArrayException

Thrown if the variable is not an array

Example

```
try {
int i[] = getIntArray();
}catch(IONNotAnArrayException e){
   system.err.println("Variable is not an array");
}
```

getShort()

The getShort() method returns the short value of the variable. If the value is not of type short (IDL type int), the scalar value is converted to a short.

Syntax

```
short s = getShort()
```

Return Value

The method returns the short value of the variable.

Arguments

None.

Exceptions

IONIsAnArrayException

Thrown if the variable contains an array value

NumberFormatException

Thrown if the variable is a string that cannot be converted.

Example

```
try {
short s = getShort();
```

```
}catch(IONIsAnArrayException e){
   system.err.println("Variable is an array");
}catch(NumberFormatException e){
   system.err.println("String Cannot be converted");
}
```

getShortArray()

The getShortArray() method returns the short array value of the variable. If the value is not of type short, a conversion is attempted.

Syntax

```
short s[] = getShortArray()
```

Return Value

The method returns the short array value of the variable

Arguments

None.

Exceptions

IONNotAnArrayException

Thrown if the variable is not an array

Example

```
try {
short s[] = getShortArray();
}catch(IONNotAnArrayException e){
   system.err.println("Variable is not an array");
}
```

getString()

The getString() method returns the string value of the variable. If the value is not of type string, the scalar value is converted to a string.

Syntax

```
String st = getString()
```

Return Value

The method returns the string value of the variable.

Arguments

None.

Exceptions

IONIsAnArrayException

Thrown if the variable contains an array value

Example

```
try {
String st = getString();
}catch(IONIsAnArrayException e){
   system.err.println("Variable is an array");
}
```

getStringArray()

The getStringArray() method returns the string array value of the variable. If the value is not of type string, a conversion is attempted.

Syntax

```
String st[] = getStringArray()
```

Return Value

The method returns the string array value of the variable

Arguments

None.

Exceptions

IONNotAnArrayException

Thrown if the variable is not an array

Example

```
try {
String st[] = getStringArray();
}catch(IONNotAnArrayException e){
   system.err.println("Variable is not an array");
}
```

isArray()

Use the isArray() method to determine if the value of the variable is an array.

Syntax

```
boolean bisArray = isArray()
```

Return Value

This method returns true if the variable is an array and false if the variable is not.

Arguments

None.

Exceptions

None

Example

```
boolean bIsArray = isArray();
```

toString()

The toString() method returns a string representation of the variables value.

Syntax

```
String st = toString()
```

Return Value

A string that represents the value of the variable. The string is in a format that can be understood by IDL.

Arguments

None.

Exceptions

None.

Example

```
String s = toString();
```

type()

The type() method returns the type of the value the variable contains. This return value is one of the constant type codes which are a part of this object.

Syntax

```
int typecode = type()
```

Return Value

This function returns the type code of the variable.

Arguments

None

Exceptions

None

Example

```
int typeCode = type();
```

IONWindow

The IONWindow class extends the Java Frame class to let the ION windows have access to the top-level window events. The windows need an IONWindowingClient registered with them as a callback object for 'destroy' events.

Class Declaration

```
public class IONWindow extends Frame
```

Methods

- **IONWindow()**
  Construct an object of the IONWindow class
- **setId()**
  Set the ID the owner uses to identify this window.
- **setOwner()**
  Tell the window who to call to destroy the window

IONWindow()

The IONWindow() method constructs an IONWindow object.

Syntax

```
IONWindow(String title)
```

Arguments

title

The title of the frame

Exceptions

None.

setId()

Use the setID() method to set the ID that the owner uses to identify the window.

Syntax

```
setId(int id)
```

Arguments

id

The id of the window.

Exceptions

None.

setOwner()

Use the setOwner() method to set the owner of the window.

Syntax

```
setOwner( IONWindowingClient owner)
```

Arguments

owner

The owner of the frame

Exceptions

None

IONWindowingClient

The IONWindowingClient class provides mechanisms to handle the processing of the windowing commands that are part of an IDL Direct graphics driver. This includes the creation, deletion, showing, hiding and iconization of windows on the client.

Class Declaration

```
public class IONWindowingClient extends IONGraphicsClient
```

Methods

- **IONWindowingClient()**
  Construct an object of the IONWindowingClient class.
- **connect()**
  Connect to the server.
- **createPixmap()**
  Creates an offscreen drawing area.
- **createWindow()**
  Creates a window on the client.
- **deleteWindow()**
  Deletes a given window or pixmap.
- **iconizeWindow()**
  Used to iconize/restore a window.
- **isPixmap()**
  Returns true if window is a pixmap
- **isWindow()**
  Returns true if window is a window.
- **showWindow()**
  Used to show/hide a window.

IONWindowingClient()

Use the IONWindowingClient() method to construct an IONWindowingClient object. This initializes the object. The connect method (from IONGraphicsClient) must be called to establish a connection between the client and the server.

Syntax

```
IONWindowingClient(Component comp)
```

Arguments

comp

A Java AWT Component that is used to referance the display being used for the graphics. This is needed for creating offscreen images.

Exceptions

None

connect()

The connect() method establishes a connection between the client and the ION Server. The client and the server make validity checks and the communication protocol is established.

Syntax

```
connect(String hostname [, int portNumber])
```

Arguments

hostname

The name of the host that the ION Server is running on. If the class is being created as part of a Java applet, most web browsers require that the host name be the same host that the applet is being served from.

portNumber

The port number to use when connecting to the ION Server. If this number is not provided the default port number is used.

Exceptions

IOException

A network IO error detected

UnknownHostException

The given hostname is unknown

IONLicenseException

An ION license could not be obtained

createPixmap()

The createPixmap() method creates an off screen drawing area of the given size, makes the drawing area the current destination for graphics output and returns the IDL window index of the new drawing area.

Syntax

```
createPixmap(int xsize, int ysize)
createPixmap(int index, int xsize, int ysize)
```

Return Value

This method returns the IDL window index of the newly created offscreen drawing area.

Arguments

xsize

The width in pixels of the pixmap to be created.

ysize

The height in pixels of the pixmap to be created.

index

The desired IDL index of the new window.

Exceptions

None

Example

```
int index = createPixmap( xsize, ysize);
int index = createPixmap( index, xsize, ysize);
```

createWindow()

The createWindow() method creates a drawing area of the given size, places that area in it's own window frame, make the window the current destination for graphics output and returns the IDL window index of the new window. If a title is not specified, the default IDL windowing convention is used (IDL 0, IDL 1, ...).

Syntax

```
createWindow(int xsize, int ysize)
createWindow(int xsize, int ysize, String title)
createWindow(int index, int xsize, int ysize)
createWindow(int index, int xsize, int ysize, String title)
```

Return Value

This method returns the IDL window index of the newly created window.

Arguments

xsize

The width in pixels of the window to be created.

ysize

The height in pixels of the window to be created.

title

The title of the window to be created.

index

The desired IDL window index of the window.

Exceptions

None

Example

```
int index = createWindow( xsize, ysize);
int index = createWindow( xsize, ysize, title);
int index = createWindow( index, xsize, ysize);
int index = createWindow( index, xsize, ysize, title);
```

deleteWindow()

Use the deleteWindow() method to delete the window/pixmap that is referenced by the given IDL Window index.

Syntax

```
deleteWindow(int index)
```

Arguments

index

The IDL Window index of the window/pixmap to destroy.

Exceptions

None

iconizeWindow()

Use the iconizeWindow() method to iconize and restore an IDL Window. This method has no effect on pixmap drawables.

Syntax

```
iconizeWindow(int index, boolean iconize)
```

Arguments

index

The IDL Window index of the window to iconize

iconize

Flag used indicate if the window should be iconized or restored.

Exceptions

None

isPixmap()

The isPixmap() method returns true if the window is a pixmap.

Syntax

```
isPixmap()
```

Arguments

None

Exceptions

None.

Example

```
boolean b = isPixmap();
```

isWindow()

The isWindow() method returns true if the window is a *window*.

Syntax

```
isWindow()
```

Arguments

None

Exceptions

None.

Example

```
boolean b = isWindow();
```

showWindow()

Use the showWindow() method to raise or lower the Z order of the given window. This method has no effect on pixmap windows.

Syntax

```
showWindow(int index, boolean show)
```

Arguments

index

The IDL Window index of the window to iconize

show

Flag used indicate if the window should be shown or hidden.

Exceptions

None

Chapter 8

Troubleshooting

The following topics are covered in this chapter:

Using ION applets over the World Wide Web and requires interaction between your web server, the ION Server, and IDL. It is beyond the scope of this manual to discuss problems with your web server setup; the following are some possible ION Server and IDL problems you may encounter.

If your applet fails to function properly, always check the Java console (Netscape browsers) or the Java log (Microsoft Internet Explorer browsers) for clues. (Microsoft Internet Explorer users should enable Java logging; the log file is named `javalog.txt` and is located in the `Java` subdirectory of the `Windows` directory. It is also helpful to set the ION applet Debug Mode (see page 45); this allows you to check the IDL command log output for errors.

Enable Java in Your Browser

Most web browsers include a setting that enables the use of Java applets in HTML pages. Make sure your browser is configured to allow Java applets to load. In Microsoft's Internet Explorer, the setting is in the "Active Content" section of the "Security" tab in the Options dialog. In Netscape's Navigator, the setting is in the "Advanced" section of the Preferences dialog.

File Permissions

The ION Daemon runs with the user and group ID of the user who started it. This means that the daemon will have the same file access permissions as that user. While it is not necessary to start the ION Daemon as a particularly privileged user, make sure that the access permissions for the ION class files and any class files you create are such that the ION Daemon has read permission.

If your applet does not run and the Java Console shows something like the following:

```
# Applet exception: class myApplet not found
```

where you know that the `myApplet.class` file exists and is located in the designated place, you may have a file permissions problem.

Starting the ION Service

On Windows NT systems, only users with the proper permissions are allowed to start and stop the ION service using the Services Control Panel. Since Administrator privileges are required to install ION, Administrator privileges are also required to start and stop the ION service.

Location of Class Files

If you encounter an error that looks like:

```
Applet xxx can't start: class xxx not found
```

in the message area of your browser or in the Java Console, check to make sure that the ION package (the directory hierarchy `com/rsi/ion/*`) or the appropriate ION archive file is located either in the same directory as the HTML page that contains the applet or in the directory specified by the CODEBASE attribute of the APPLET tag. See "Locating the Class Files for use by ION Applets" on page 39 and "Supporting Java Archive Files" on page 39 for details.

Location of IDL .pro Files

If you call user-written IDL routines from an applet, make sure that the `.pro` files are located in IDL's path. You can do this either by placing the directory that contains your `.pro` files in the path specified by the IDL_PATH environment variable, by starting the IDL Development Environment and adding your directory in the "Path" tab of the Preferences dialog, or by explicitly altering the value of the IDL system variable !PATH within your applet code.

If ION attempts to compile and run a `.pro` file that is not in the path, no output will be generated but no error will be displayed. The best way to catch errors like this is to enable the ION applet Debug Mode (see page 45) and check the IDL command log output.

Browser Timeout on Error

If you do encounter an error when running a Java applet, some browsers' Java virtual machines will "hang," requiring you to shut down and restart the browser. It is generally a good idea to restart your browser after a Java error.

When the error is in an ION applet, there is a chance that the connection to the ION Server is still active when you close your browser. In this case, your browser may not start again immediately; it will wait for the ION socket connection to time out before shutting down and allowing you to start the browser again.

On Unix systems, you can use the kill command to prematurely kill the browser process and close the socket connection. On Windows NT systems, use the "Processes" tab of the Task Manager dialog to end a browser process. If you do not manually kill the browser process, the socket connection will automatically time out in 60 seconds.

Index

A

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z